TRANSLATION OF A QUEUING PROBLEM DESCRIPTION INTO GPSS-LIKE TABLES

John Henry Rickelman

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TRANSLATION OF A QUEUING PROBLEM DESCRIPTION INTO GPSS-LIKE TABLES

by

John Henry Rickelman

Thesis Advisor: G. E. Heidorn

June 1972

Translation of a Queuing Problem Description

Into GPSS-Like Tables

by

John Henry Rickelman
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1960

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1972

ABSTRACT

A system called NLPQ is being developed at the Naval Postgraduate School as part of a research project in natural language man-machine communication. The basic part of the system consists of a rule language and programs to compile and execute rules. NLPQ currently allows a user to enter an English text description of a simple queuing problem at a time sharing terminal, have the computer construct an Internal Problem Description, and then produce an equivalent English text description and a GPSS simulation program to solve the problem.

Recently, a FORTRAN routine for performing a GPSS-like simulation was incorporated into the system. This thesis reports on the development of another set of rules for NLPQ to translate an Internal Problem Description into an array of information required by this FORTRAN routine to perform the simulation.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

In present times, computer technology more and more embraces the concept of distribution of on-line computing power to a large number of users via a complex system of data communications lines and sophisticated terminals. These systems already play an important part in the scientific, educational, military, and business communities by providing for source data input and response via typewriter, crt, or special purpose terminals. However, the users must learn one or more special computer languages in order to use one of these systems, and even then are limited to those tasks for which the language was designed.

One way to surmount the problems of special training and limited capabilities of computer languages is to permit man-computer communication in the natural language of the user. Research in the area of natural language communication with a computer has generally focused on one major stumbling block, that of translating natural language statements into a precise data structure which may be used by the computer to accomplish desired tasks. Present day work centers about various language theories. One theory possessing considerable merit is that of Stratificational Grammar by Sydney Lamb [1].

A project to investigate methods and techniques of a natural language system is being conducted at the Naval

Postgraduate School in Monterey, California [2,3]. The present objective of this project is to investigate a particular application of natural language processing, that of describing a queuing problem to a computer in English text and having it execute a simulation program to solve the problem. The system being developed to meet the project objective is called NLPQ and it utilizes a more general system, called NLP, both of which will be described in a later section.

A. BACKGROUND

The basic components of the NLP system were part of the initial work on the project and consist of a system monitor made up of a number of FORTRAN programs and a rule language whose statements are compiled and executed by the FORTRAN programs. The system runs on the IBM 360/67-CP/CMS time sharing system at the Naval Postgraduate School. Since the initial work on NLP, development of the NLPQ system has resulted in improvements to NLP and a series of modules written in the NLP rule language. The names of the modules and the functions they perform are as follows:

1. English decoding rules- translates English text representing a queuing problem into an internal structure known as the Internal Problem Description (IPD)[2,3,4].

2. English encoding rules- processes the IPD to produce an English text description of the IPD which may be compared against the input English text to check for accuracy [5].

3. GPSS encoding rules- processes the IPD to produce a GPSS program [5,6,7].

4. Massager rules- inspect an IPD structure and add information that might be assumed by a person that has some knowledge of the problem [8].

5. Interrogator rules- allows a user to enter a queuing problem in a question-answer mode and also inspects an IPD for missing or erroneous information that might result in an incomplete or incorrect GPSS program [8].

The end-product of the above modules is a GPSS program, which has to be run separately to perform the simulation. A recent addition to NLPQ is a FORTRAN simulation routine [9] whose algorithm is basically the same as that of the GPSS simulator [7]. This routine takes its input from a one dimensional array, called the X-Vector, structured in the same manner as the "internal tables" of the GPSS simulator. In order for NLPQ to utilize the simulator for performing a simulation "on-line", it then was necessary to develop a translational function capable of mapping the IPD representing the queuing problem into the table allocations of the X-Vector.

B. THESIS OBJECTIVE

The objective of this thesis, then, was to develop a set of encoding rules for producing the X-Vector from an Internal Problem Description. Because of the basic similarities between the X-Vector and a GPSS program, it was decided to write one set of rules which could produce both the X-Vector and/or a GPSS program, thereby obsoleting the original GPSS encoding rules.

C. ORGANIZATION OF THE THESIS

This thesis is divided into five sections. Section II presents a description of pertinent portions of NLP and NLPQ. Section III discusses a sample problem. Section IV presents considerations involved in accomplishing the objective of this thesis and describes a set of encoding rules which fulfill the objectives. Section V presents conclusions and recommendations.

II. DESCRIPTION OF NLPQ

The IPD to X-Vector translational function is a module constructed in the NLP rule language utilizing features of both the physical and logical information structure of the IPD to accomplish its task. It is necessary, therefore, to describe these features in order to better understand the description of the module developed to accomplish the objective of this thesis. The cell array, being the nucleus of the entire system, is discussed first.

A. CELL ARRAY

The basic physical element of the information structure is called a "cell". On the IBM 360/67 the cell consists of 8 bytes, i.e., 64 bit positions. Most cells are divided into four fields of two bytes each. Bytes 1 and 2 are called the TYPE, bytes 3 and 4 are called the ATTR (attribute), bytes 5 and 6 are called the ADDR (address), and bytes 7 and 8 are called the LINK. The TYPE field contains a number (0,1,2,3) that specifies the type of cell. The ATTR field specifies the attribute number of the cell. ADDR is a number that stands by itself as a numeric value or is a pointer to another cell, depending on the value of type. LINK is a pointer to the next cell in a "list". In a cell of type 0 the value of ADDR contains numeric information; in a type 1 cell, ADDR is a pointer to another cell that contains bit oriented information or the EBCDIC representation of a

characters; in a type 2 cell, ADDR is a pointer to a "local list" of cells; and in a type 3 cell, ADDR is a pointer to a "record".

A "list" is a group of cells connected through the LINK field, in which the last cell on the list has a LINK of zero. Both a "local list" and a "record" may be called a "list". However, a "record" has special features in that the first cell is of type 0 and the ADDR field contains a counter that indicates how many type 3 cells located elsewhere in the cell structure point to this record.

The nucleus of the IPD logical structure is the previously defined "record". A record represents some unique item such as a book, ship, action, thought, word, etc.. This record, then, represents some "entity" which has certain "attributes". Physically, each cell of the list that represents a record, holds the value of an attribute of that record. Each attribute has a number located in the ATTR field of the cell, and the cells of a record are linked in numerical sequence by attribute number. There are facilities in the system for associating a name with an attribute number. This may be done explicitly in a declaration or implicitly in the process of writing rules for an application.

A special type of attribute is called an indicator and may be used when an attribute value may be specified in a bit position as 0 or 1. The individual bit positions of attribute 2 are used to hold these on/off value attributes. The indicator name and position of these on/off attributes are associated explicitly in a declaration. Other declared

attributes are attribute 1, named SUP, which points to another record considered to be a superset of the record of which SUP is an attribute, and attribute 10, called NAME, which contains the EBCDIC representation of a record's name. A record with an attribute 10 is considered to be a "named record". A reference to a named record for the entity "ship" would be in the form 'SHIP'. Named records with specified attributes may be declared in a similar manner to attributes and indicators.

The logical structure of the IPD becomes, basically, a group of records connected in some logical manner by pointers in the ATTR and ADDR fields of the attribute cells. The actual form of the structure depends both on predetermined associations created by declared indicators, attributes, and named records, and on dynamically created associations that come about when the decoding rules are applied to input text.

B. INTERNAL PROBLEM DESCRIPTION (IPD)

The queuing problem that NLPQ undertakes to solve may be thought of as consisting of a number of unique items called entities. Entities are unique in the sense that each may be distinguished from all other items or "things" involved in the problem. Entities are further divided into two groups called physical entities such as a truck, gas station, ship, or pier, and abstract entities such as an action or function. Physical entities are then grouped into stationary entities and mobile entities. A general description of the flow in

a queuing problem is that the mobile entities engage in actions at the various stationary entities.

A problem description in English could be divided into the parts mentioned above. The basic structure of the problem would center on a sequence of actions being performed on or by an object at some location. The location is a stationary entity, the object is a mobile entity, and the action is an abstract entity. Once the mobile entity completes a specific action it moves on to a following action at the same or a different stationary entity depending on local conditions at the stationary entity.

In the IPD, entities are represented by records which consist of attributes and their values. Attribute values need not be numeric and may be pointers to other records depending on their complexity. The heart of the IPD is a structure representing the various actions that would take place in the problem. For most problems those actions are: The arrival of a mobile entity, one or more servicings of the mobile entity and, finally, departure. Depending on the problem description, multiple paths may exist between arrival and departure. Records, representing the actions, are linked by pointers to form the structure. These action records possess certain attributes, which point to other records representing mobile entities and stationary entities associated with that particular action. Other records in the IPD represent functions, including distributions, necessary to furnish a complete description of the problem. In order to be able to locate specific records to make attribute

or value changes, lists of pointers to the various entity records are also maintained in the IPD. There are lists of actions, mobile entities, stationary entities, distributions, and successor descriptors. A successor descriptor is a record giving information about which action should occur next. Pointers to these list records are maintained in a special record called MEMORY which also possesses other attributes containing specific items of information necessary to the problem. In the GPSS structure, upon which NLPQ is based, transactions are equivalent to mobile entities and a series of what are called executable blocks are equivalent to an action.

C. DECODING AND ENCODING RULES

In NLPQ the processing of input text to produce an IPD or its reverse, converting an IPD to some output text or structure, is performed within the framework of the Stratificational Grammar theory. This theory identifies three levels of language structure as being morphological, lexological, and semological. In general, it can be said that considerations at the morphological level are highly dependent on the particular language, while semological elements are concerned with relationships or meanings and are relatively language-independent. The lexology represents a middle ground, but is still language-dependent. Consequently, in the decoding process, morphological and lexological rules are applied first in processing input text, and semological rules develop the structure representing the meaning of the

text, (i.e., the IPD). The encoding process applies semo-logical rules to the IPD first, and then rules at the lexo-logical and morphological levels to produce English text, GPSS code or the specific elements of the X-Vector.

Details of the decoding process may be found in [2] and [3], and will not be included in this section as they are not germain to this thesis. Because the encoding process is applied to translate the IPD to the X-Vector, the encoding rule processing procedure will be discussed here.

The nature of the encoding process is such that at every step of the IPD-to-output translation there are certain temporary records, called segments, created. These temporary records are processed by applying rules until an output occurs. At any step in the process any segment may be identified by a unique name known as a segment type name. At the time of compilation, a permanent record known as a segment type record is created. This record has a name attribute containing the name of the segment type and another attribute pointing to a list of encoding rules that have that name on the left of the arrow. The format of the encoding rules is:

SEGMENT TYPE (CONDITION 1, CONDITION 2--) -->
SEGMENT TYPE (ACTION 1, ACTION 2,--)
SEGMENT TYPE (ACTION 1, ACTION 2,--)

A rule consists of a left side and a right side, separated by the symbol -->. The right side may consist of more than one segment type. When a segment identified by the segment type on the left is encountered and the conditions in

parentheses are met, segments identified by the segment types on the right are created and the actions in parentheses performed. These segments in turn are processed by rules with their segment type identifier on the left. Typical conditions on the left are the presence or absence of an indicator or attribute in the segment, satisfaction of some relationship between two values, or similar tests on attributes or indicators of some other record. Actions performed may be such things as addition or deletion of attributes or indicators of a record or copying of all or part of a record into the new segment.

D. INTERROGATOR/MASSAGER

Details of these two modules are contained in [8] and only a brief explanation is provided here for an overall understanding of the system. The object of the Massager is to provide values that might logically be assumed by a person knowledgeable about the queuing problem. All assumptions are expected to be non-controversial. The interrogator operates in a question-answer mode and permits entry of the problem piecemeal; or it may be used to inspect the IPD once the problem has been described, to detect erroneous or incomplete information. The interrogator is involved automatically by the system after decoding and massaging have occurred and before encoding takes place.

E. FORTRAN SIMULATION ROUTINE

This GPSS-like simulation routine is completely described in [9], but a short description will be provided here. The

routine takes its input from a single dimensioned array, known as the X-Vector, and is initialized by the calling program. The structure of the information in the array is basically that of the GPSS internal tables. The calling program must build in the array allocations of groups of contiguous elements that constitute GPSS entities. These entities consist of STORAGES, QUEUES, TABLES, FUNCTIONS, VARIABLES, SAVEVALUES, and BLOCKS.

STORAGE and QUEUE allocations are made for the stationary entities of the problem. TABLE allocations occur for the problem's mobile entities. FUNCTION allocations evolve from the distributions and problem conditional paths. VARIABLES occur as needed, such as to describe a normal distribution. BLOCKS evolve from the various problem actions that occur.

These allocations may be placed anywhere in the X-Vector except for the first 32 elements. Eleven of the first 32 elements are reserved for certain attributes needed for all problems, and the remaining 21 contain information about what is in the rest of the array. Two locations for each entity in the first 32 elements contain the number of entities of that type and a pointer to a directory that points to each allocation for that type. The X-Vector for a sample problem will be shown in the next section.

F. ROUTINES

Basic to the construction of the X-Vector is the concept of rule-FORTRAN communication. This is accomplished through the use of "routines". Routines make it possible to

processing that cannot be specified in the NLP rule language. In order to do this, these routine names and identifying numbers must be declared prior to compiling the rules, and the actual routines must be incorporated into the FORTRAN program. For this thesis routines were written to store values in the X-Vector and to retrieve values from the X-Vector. A listing of these routines appears in Appendix A.

III. SAMPLE PROBLEM

Figure 1, extracted from Ref. 2, is an example of a queuing simulation problem. This English description was produced by encoding an Internal Problem Description in the manner described in Ref. 2.

The GPSS/X-Vector rules were applied to the same IPD, and the results are listed in Figure 2. Each GPSS statement is followed by its equivalent in the X-Vector. Where three numbers are listed, they give the X-Vector element number, the value of the left half of the element, and the value of the right half. Where there are only two numbers, they give the X-Vector element number and its decimal value.

As can be seen, first a SIMULATE block is put out with no corresponding X-Vector allocation. Then an RMULT statement with 8 seeds is emitted. Next, the seeds are entered in predesignated elements 3 through 10 of the X-Vector. Then GPSS EQU statements for stationary entities STAT1 and PUMP2 are issued. Corresponding X-Vector STORAGE and QUEUE allocations are made. GPSS TABLE definitions come next, with TABLE allocations made in the X-Vector. The following four FUNCTIONS and FVARIABLE are handled in the same manner. Following these are block allocations, beginning with a GENERATE block, and ending with a TERMINATE block in the timing loop. Next are the START and END control statements. Then the entity directories are set up and pointers placed in the appropriate locations within the first 32 elements. Finally, the

appropriate block numbers are backstuffed into the third argument of a text block allocation and the second argument of a transfer allocation.

Figure 3 shows the layout of an X-Vector, with some information from the sample problem. First are the STORAGE and QUEUE allocations. A QUEUE allocation follows every STORAGE allocation. Then come the exponential and normal FUNCTION allocations, if required, followed by other FUNCTION allocations. VARIABLE allocations come next, followed by BLOCK allocations. Then come the directories which are in the same order as the allocations.

The following example describes the use of the directory to locate an allocation. The contents of element 12 specifies that there are 2 storages and element 13 contains a pointer to the STORAGE directory, in this case, element 336, one less than the location of the first element in the directory. The two elements in the directory (337 and 338) contain pointers to the allocations, which begin in elements 33 and 51. The elements of each allocation are filled by the calling program or reserved for use by the simulator to collect statistics during simulation. A complete description of the significance of each allocation element may be found in Ref. 9. Those elements filled by the GPSS/X-Vector encoding rules, will be described in the next section.

Another sample problem is given in Appendix B. This is the "Harbor Problem" which originally appeared in Refs. 5 and

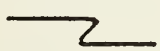
6. In the appendix the named records defining the IPD are given first, followed by the encoded English problem description. Then the output produced by the GPSS/X-Vector encoding rules is given.

THE VEHICLES ARRIVE AT THE STATION. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 8 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES. 75 PERCENT OF THE VEHICLES ARE CARS, AND THE REST ARE TRUCKS. AFTER ARRIVING AT THE STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS LESS THAN 2, THE VEHICLE WILL BE SERVICED AT THE PUMP IN THE STATION. OTHERWISE, THE VEHICLE WILL LEAVE THE STATION. THE TIME FOR THE VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 5 MINUTES FOR THE CARS, AND 9 MINUTES FOR THE TRUCKS. AFTER BEING SERVICED AT THE PUMP IN THE STATION, THE VEHICLES LEAVE THE STATION.

THE SIMULATION IS TO BE RUN FOR 8 HOURS, USING A BASIC TIME UNIT OF 30 SECONDS.

FIGURE 1 - ENGLISH DESCRIPTION OF THE SAMPLE PROBLEM

```
        SIMULATE
        RMULT       277,423,815,121,655,531,999,813
    3   0   277
    4   0   423
    5   0   815
    6   0   121
    7   0   655
    8   0   531
    9   0   999
    10   0   813
STAT1   EQU         1,F,Q
    34   0   1
    42   0   4069
    50   4069   0
PUMP2   EQU         2,F,Q
    52   0   1
    60   0   4577
    68   4577   0
CAR2    EQU         2,T
2       TABLE       M1,1,1,2
    74   0   1
    76   0   2
    77   0   1
TRUC3   EQU         3,T
3       TABLE       M1,1,1,2
    85   0   1
    87   0   2
    88   0   1
1       FUNCTION    RN1,C24
    91   -26   1
    92   -23   6141
    94   -25   0
    95   1   24
0.0,0.0/.100,.104/.200,.222/.300,.355/
.400,.509/.500,.690/.600,.915/.700,1.200/
.750,1.390/.800,1.600/.840,1.830/.882,2.120/
.900,2.300/.920,2.520/.940,2.810/.950,2.990/
.960,3.200/.970,3.500/.980,3.900/.990,4.600/
.995,5.300/.998,6.200/.999,7.000/1.000,8.000/
    96   1   1
    97   0.0
    121   0.0
    98   .100
    122   .104
```



```
    119   .999
    143   7.000
    120   1.000
    144   8.000
2       FUNCTION    RN2,C29
    169   -26   2
    170   -23   4421
    172   -25   0
    173   1   29
0.0,-3.000/.012,-2.250/.027,-1.930/.043,-1.720/
.062,-1.540/.084,-1.380/.104,-1.260/.131,-1.120/
.158,-1.000/.187,-.890/.230,-.740/.267,-.620/
.334,-.430/.432,-.170/.500,0.0/.568,.170/
.666,.430/.732,.620/.770,.740/.813,.890/
.841,1.000/.869,1.120/.896,1.260/.916,1.380/
.938,1.540/.957,1.720/.973,1.930/.988,2.250/
1.000,3.000/
    174   1   1
    175   0.0
    204   -3.000
```

FIGURE 2 - GPSS PROGRAM AND X-VECTOR FOR THE SAMPLE PROBLEM

```
     176  .012
     205  -2.250
          ~~~~2~~~~

     202  .988
     231  2.250
     203  1.000
     232  3.000
 3      FUNCTION   P1,D2
     262  -1  1
     263  -23  6343
     265  -25  0
     266  2  2
CAR2,10/TRUC3,18/
     267  0  0
     268  0  2
     270  0  10
     269  0  3
     271  0  18
 4      FUNCTION   RN3,D2
     272  -26  3
     273  -23  6197
     275  -25  0
     276  2  2
.750,CAR2/1.000,TRUC3/
     277  1  0
     278  .750
     280  0  2
     279  1.000
     281  0  3
 1      FVARIABLE  16+4*FN2
     282  0  16
     283  0  4
     284  -22  2
     285  -30  3
     286  -30  1
     287  -25  0
*
*       THE VEHICLES ARRIVE AT THE STATION.
        GENERATE   V1
     288  1  4351
     289  -24  1
     290  -33  3811
        ASSIGN     1,FN4
     291  4  4351
     292  0  1
     293  -22  4
     294  -33  2
        TEST L     Q$PUMP2,2,ACT2
     295  11  4351
     296  -12  2
     297  0  2
     298  0  4090
     299  -33  2
        TRANSFER   ,ACT3
     300  3  4351
     302  0  3824
     303  -33  0
*
*       THE VEHICLES LEAVE THE STATION.
 ACT2  TABULATE   P1
     304  12  4090
     305  -3  1
     306  -1  1
     307  0  -1
     308  -33  0
        TERMINATE
     309  10  4090
```

FIGURE 2 (CONTINUED)

```
    310  -33   0
*
*       THE VEHICLES ARE SERVICED AT THE PUMP.
 ACT3   QUEUE         PUMP2
    311   8   3824
    312   0   2
    313   -33   0
        SEIZE         PUMP2
    314   6   3824
    315   0   2
    316   -33   0
        DEPART        PUMP2
    317   9   3824
    318   0   2
    319   -33   0
        ADVANCE       FN3,FN1
    320   2   3824
    321   -22   3
    322   -22   1
    323   -33   0
        RELEASE       PUMP2
    324   7   3824
    325   0   2
    326   -33   0
        TRANSFER      ,ACT2
    327   3   3824
    329   0   4090
    330   -33   0
*
*       TIMING LOOP
        GENERATE      960
    331   1   0
    332   0   960
    333   -33   0
        TERMINATE     1
    334   10   0
    335   0   1
    336   -33   0
        START         1
    1   0   1
    2   0   1
        END
    13   0   336
    12   0   2
    337   0   32
    338   0   50
    15   0   338
    14   0   2
    339   0   42
    340   0   60
    21   0   340
    20   0   3
    341   0   0
    342   0   68
    343   0   79
    17   0   343
    16   0   4
    344   0   90
    345   0   168
    346   0   261
    347   0   271
    32   0   347
    31   0   1
    348   0   281
    19   0   348
    18   0   14
    349   0   287
    350   0   290
    351   0   294
```

FIGURE 2 (CONTINUED)

```
352   0   299
353   0   303
354   0   308
355   0   310
356   0   313
357   0   316
358   0   319
359   0   323
360   0   326
361   0   330
362   0   333
24   0   3
25   0   362
298   0   5
302   0   7
329   0   5
```

FIGURE 2 (CONTINUED)

1	MODE 1	29	LAST TRANS. FEC. PTR. 0	282	VARIABLE ALLOCATION		
2	TERMINATION COUNT 1	30	ERROR CODE 0	288	BLOCK ALLOCATIONS		
3	SEED1 277	31	NUMBER OF VARIABLES 1				
4	SEED2 423	32	VARIABLE DIR. PTR. 347				
5	SEED3 815	33	STORAGE ALLOCATION (STAT1)	337	STORAGE 32 DIRECTORY 50		
6	SEED4 121						
7	SEED5 655						
8	SEED6 531						
9	SEED7 999						
10	SEED8 813	43	QUEUE ALLOCATION (STAT1)	339	QUEUE 42 DIRECTORY 60		
11	CLOCK TIME 0						
12	NUMBER OF STORAGES 2						
13	STORAGE DIR. PTR. 336	51	STORAGE ALLOCATION (PUMP2)	341	TABLE 0 DIRECTORY 68 79		
14	NUMBER OF QUEUES 2						
15	QUEUE DIR. PTR. 338	61	QUEUE ALLOCATION (PUMP2)				
16	NUMBER OF FUNCTIONS 4			344	FUNCTION DIRECTORY		
17	FUNCTION DIR. PTR. 343	69	TABLE 2 ALLOCATION				
18	NUMBER OF BLOCKS 14						
19	BLOCK DIR. PTR. 348	80	TABLE 3 ALLOCATION	348	VARIABLE DIRECTORY		
20	NUMBER OF TABLES 3						
21	TABLE DIR. PTR. 340	91	EXPON FUNCTION ALLOCATION	349	BLOCK DIRECTORY		
22	NUMBER OF SAVEVALUES 0						
23	SAVEVALUE DIR. PTR. 0						
24	NUMBER OF PARAMETERS 3	169	NORMAL FUNCTION ALLOCATIONS	363	TRANSACTION ALLOCATIONS		
25	TRANSACTION POINTER 362						
26	FIRST TRANS. CEC. PTR. 0						
27	LAST TRANS. CEC. PTR. 0	262	ADDITIONAL FUNCTION ALLOCATIONS				
28	FIRST TRANS. FEC. PTR. 0						

Figure 3. X-Vector Layout

IV. GPSS/X-VECTOR ENCODING RULES

This section will discuss the logic flow and details of the GPSS/X-Vector encoding rules. An overview will be presented first, followed by a detailed description of the rules. Appendix C contains a listing of the rules, each of which is numbered for ease of reference. Appendix D summarizes the various segment types and the attributes they normally hold.

A. OVERVIEW OF THE GPSS/X-VECTOR RULES

APPENDIX C is divided into the following six parts:

1. Indicator, Attribute, Routine, and Named Record declarations
2. Semology for encoding GPSS/X-Vector
3. Lexology for encoding GPSS
4. Morphology for encoding GPSS
5. Lexology for encoding X-Vector
6. Morphology for encoding X-Vector

Indicator, Attribute, and Routine declarations establish records which associate a symbolic name with a number to be used in the rule processing phase. Named Record definitions create records with a NAME attribute value as designated in the left column and other attribute value pairs as shown in parentheses. The form 'NAME' in parentheses sets the value of attribute 1, the SUP attribute, to point to the specified named record.

The structure of the GPSS/X-Vector rules is based on the format of a GPSS statement consisting of the following four fields:

Label field-identifies or points to a particular GPSS block. The attribute LABL represents this field.

Operation field- identifies the nature of an action or definition statement. The SUP attribute of a STMNT segment represents this field.

Mod field- amplifies the operation field. The attribute MOD of segment STMNT represents this field.

Argument field- may contain from one to seven separate arguments called Standard Numerical Attributes, depending on the nature of the operation field. The attributes ARGA through ARGH represent this field.

The rules are divided into five groups. There is one set of semological rules, whose function is to examine the IPD and produce STMNT segments, roughly corresponding to the statements of a GPSS program. The GPSS lexology produces other segments with the information from the STMNT segments in an order appropriate for a GPSS program. The GPSS morphology actually produces the output from these segments. Similarly to the GPSS lexology, the X-Vector lexology also produces other segments with the information from the STMNT segments, but in an order appropriate for the X-Vector. The

X-Vector morphology actually places the information in the X-Vector. The following discussion will provide an overall view of the logic flow involved in the processing.

The creation of the segment GPSSPROG provides entry into the GPSS/X-Vector rules at the semological level. From GPSSPROG, two processing segments, INITIALGPSS and SETAGL are created to preset the values of various attributes. The rest of the segments created by GPSSPROG lead ultimately to a STMNT segment with, as appropriate, SUP, LABL, MOD, and ARGA through ARGH attributes. Preprocessing of certain STMNT segments as determined by the conditions on the left is done in rules 41 through 45, but ultimately all STMNT segments are processed by rule 46 and passed to the GPSS and X-Vector lexological level rules by the creation of segments GSTMNT and XSTMNT. The setting of indicator attribute NOCOPY prevents production of GPSS code or the X-Vector unless the indicator attributes GPSSSW of MEMORY or XVSW of MEMORY are set. Then the segments GSTMNT or XSTMNT or both become a copy of STMNT, and attribute NOCOPY is erased.

At the GPSS lexological level the segment type GSTMNT is tested for various conditions on the left and produces segment structures that represent X-Y coordinates of a GPSS FUNCTION or comments to be inserted at each occurence of an action in the GPSS program. A GSTMNT segment type rule with no conditions on the left processes, with three exceptions, all other segments passed to this level and creates a segment structure representing the contents of the various fields

of a GPSS block. At this stage in the development of NLPQ most GPSS code will have only three arguments. However, the three exceptions mentioned above, a segment type GSTMNT with a SUP attribute of 'RMULT', 'SELECT', or 'TABLE' may have as many as seven arguments. The final segment created at this level is of type ARG and is processed at the morphological level.

GPSS morphological level rules process the ARG segment records to produce, ultimately, a segment of type OUTPUT which causes an output at the terminal or to an internal file, complete with control functions such as line skip or tab. Output may be characters from the EBCDIC value of a NAME attribute or a numerical value from the ADDR field of a type 0 cell. A determination is made by examining the TYPE field of the ARG segment. A type 0 causes creation of a NUMBER segment which goes to OUTPUT. A type 1 results in a NAME segment which goes to OUTPUT. A type 2 cell does not occur, a type 3 causes creation of a PARG segment whose subsequent rules test for various conditions on the left and create segments that output characters or numerical values to fill the argument field of a GPSS block.

X-Vector lexological rules process XSTMNT segment types created at the semological level to produce allocations of contiguous cells in the X-Vector for entities required by the FORTRAN simulation routine. At this level allocations for STORAGES, QUEUES, TABLES, FUNCTIONS and VARIABLE entities are set along with a directory and a directory locator for each entity. XSTMNT segments representing GPSS block entities

are processed by rule 127 with no condition on the left, and it indirectly creates segments of type XARGFLD to fill the elements of a BLOCK argument field with the proper Standard Numerical Attribute (SNA) or numerical value. One exception to the general structure of this level is an XSTMNT Segment with a SUP of 'RMULT' which is processed by this rule to load predesignated X-Vector elements 3 though 10 with random number seeds. Segments resulting from this level are of type XCODE, XVECTOR or XARG.

At the X-Vector morphological level, processing of an XCODE segment creates a segment of type XVECTOR which outputs numerical values to the various X-Vector allocation elements by calling upon routines. A segment of type XARG is examined for its type. A type of 0 causes creation of an XVECTOR segment, a TYPE of 1 or 2 goes to null, a TYPE of 3 creates an XPARG segment, whose rules then cause an output to load the elements of each block entity.

B. GPSS/X-VECTOR RULE EXPLANATIONS

This section will describe the rules of APPENDIX C in detail. The declaration items there will be discussed along with the rule explanations.

1. Semology for encoding GPSS/X-Vector

a. GPSSPROG (Rule 1)

Processing of this segment type creates 13 segments, 11 of which represent all of the elements required for a GPSS program. The other two, INITIALGPSS and SETAGL, perform certain initializing actions. The record MEMO is

examined to locate, in the IPD, the stationary entity list (SELIST), mobile entity list (MOBLIST), distribution list (DISTLIST), successor list(SUCLIST), and the action list (ACLIST). STMNT('SIMULATE') and STMNT(%'SEEDS') initiate action to output a SIMULATE block and an RMULT card. EXPFUNC and NORMFUNC initiate output for exponential and normal distributions, if required. TMLOOP creates the timing loop.

b. INITIALGPSS (Rule 2)

The created segments initialize the GPSS and X-Vector random number counters GRNNO of MEMORY and XRNNO of MEMORY, and attribute LR of eight named records whose value is the number of the last attribute of that record. If the indicator XVSW of MEMORY is set, i.e., its value is other than zero, a call to the routine ZEROXV is made to initialize all elements of the X-Vector to zero.

c. SELIST and STENTITY (Rules 4, 5, 8)

Rule 4- creates a copy of each stationary entity record in the IPD for further processing. The condition on the left(LC.LE.LASTREC) is common to all rules examining the various lists. The LC attribute is initially set to 11 in GPSSPROG and is incremented until it is equal to the attribute number pointing to the last entity on the list. When the condition is met, rule 5 then applies and no further processing is done, as a NULL segment is created. The attribute IPDP points to the original record in the IPD for later use in the X-Vector processing rules.

Rule 8- creates an EQUCARD and a STMNT which will cause a STORAGE definition output unless the stationary

entity is a facility. However, a STORAGE will be assigned in the X-Vector for both a STORAGE and a FACILITY. An IPDP attribute pointing to the IPD record is also included.

d. MELIST and MENTITY (Rules 6, 7, 9, 12)

Rule 6- creates MENTITY, a copy of each mobile entity record.

Rule 9- creates EQUCARD and TABLECARD.

Rule 12- creates a STMNT for a GPSS TABLE definition and an X-Vector TABLE allocation. Presently ARGA is preset to tabulate M1, the elapsed time from transaction generation to termination. Values of the attributes LOWINT, INTWIDTH, NUMINT are presently set in the Massager.

e. EQUCARD (Rule 11)

Creates a STMNT for a GPSS EQU card that equates the value of an entity's IDNAME and its IDNO. There is no equivalent X-Vector output.

f. EXPFUNC, NORMFUNC, DISTLIST, SUCLIST, FNDEF (Rules 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)

Create a FNDEF which in turn creates a STMNT segment for a FUNCTION definition.

When the queuing problem requires exponential or normal distributions, records with a SUP of 'EXPON' or 'NORMAL' are created and placed on the distribution list ('DSTRLIST'). The massager examines the distribution list and sets the indicators EXPUSED of MEMORY or NORMUSED of MEMORY when appropriate. If the indicators are present, then rules 14 and 16 create a FNDEF, a copy of the named records

'EXPONREC' or 'NORMREC', with the attribute IPDP pointing to these named records.

Rules 18 and 20- create 2 FNDEF for each distribution record and 2 successor descriptor records requiring a function definition. Both set IPDP attributes pointing to an IPD record.

Rule 22- a condition on the left, the presence of an XYLAST attribute, eliminates the processing of segments with a SUP of 'EXPON' or 'NORMAL' created by rule 1, as exponential and normal distributions were already handled by rules 14 and 16. Segments from Rule 20 that do not require a FUNCTION definition are also passed over. The first STMNT segment created produces, ultimately, a GPSS function definition and the first elements of an X-Vector function allocation. The second STMNT segment is processed by rules at lower levels to output GPSS function follower cards and X-Vector X and Y coordinate elements. The value of ARGB is a copy of the named record 'ARGREC' which is given the attributes FNTYPE(function type) and NOPTS(number of points). The values of these attributes are used when processing at lower levels to produce a character and number, such as D2, that represents argument B of a function definition.

g. DSTLIST and DISTDEF (Rules 24, 25, 26, 27)

These rules examine the distribution list for a record representing a normal distribution and issue a STMNT to define an FVARIABLE for the distribution.

h. ACTLIST, ACTT, and ACT (Rules 28 through 36)

Rule 28- creates ACTT, a copy of a specific action record from the IPD. The value of attribute IPDP of MEMORY is a pointer to the original action record.

Rule 30- creates a STMNT that is processed ultimately to output a comment card that precedes each group of GPSS blocks representing an action. The segment ACT is later processed to produce a series of STMNT segments for issue of GPSS blocks or X-Vector allocations representing the action.

Rules 31 through 38- process an action record and create a series of STMNT segments representing a standard set of executable blocks for each particular type of action.

Rule 31- represents the arrival of a mobile entity.

Rules 32 and 33- examine action records for a SUP attribute in the set 'ACTIVITY'. Rule 32 handles the situation in which the stationary entity is a STORAGE, and Rule 33 is for a FACILITY.

Rule 35- the action SEGMNT with a sup of 'LEAVE' creates segments for TABULATE, LEAVE, and TERMINATE blocks.

i. SUCSTMNT (Rules 37, 38, 39)

The records processed by these rules are a copy of the SUCC(successor) attribute of an action record. Successors may depend on some condition such as the availability of a facility or storage or the length of a line to determine which of two or more actions are to take place. The SUP of the successor record determines the attributes of the STMNT segment created. A SUCSTMNT with a SUP of 'QTYP'

produces STMNT'S for TEST and TRANSFER block. A SUP of 'FTYP' or 'STYP' produces STMNT'S for GATE and TRANSFER blocks. If none of the above conditions is met, a STMNT for a TRANSFER block is created.

j. TMLOOP (Rule 40)

The STMNT segments created here will cause production of GENERATE and TERMINATE blocks and START and END cards in GPSS. GENERATE and TERMINATE block allocations will also be made in the X-Vector. However, STMNT segments with a SUP of 'START' or 'END' cause other processing, the details of which will be given in the explanations for rules 119 and 120.

k. STMNT (Rules 41 through 46)

Rule 43- if ARGA points to a record with a SUP of 'UNIFORM' the argument field of that STMNT requires ARGA and ARGB to completely describe the uniform distribution. The argument values are taken from the MEAN and RANGE attributes of the segment being processed.

Rule 44- the same reasoning as in rule 43 applies. In this case ARGB points to a named record 'SNAREC' and attributes NAM and NUM are assigned. Their values are output through rule 77 for GPSS and rule 145 for the X-Vector.

Rule 45- tests the STMNT created by rule 38 to determine the value of MOD and the contents of the argument field for a GATE block.

Rule 46- as mentioned previously, this rule creates segments GSTMNT or XSTMNT or both if the GPSSSW of MEMORY and XVSW of MEMORY indicators are set. The same

record is sent to GPSS and X-Vector rules for processing to produce their own particular output.

2. Lexology for encoding GPSS

a. GSTMNT (Rules 47 through 51, 54)

Rule 47- if the indicator NOCOPY is set, a GPSS statement is not produced.

Rule 48- the presence of attribute XYLAST creates an XYOUT segment and initiates the output of the X-Y coordinates of a FUNCTION definition.

Rule 49- prevents the output of a STORAGE definition for a FACILITY.

Rule 50- in rule 35 ARGB of the segment was assigned the value that a GPSS TABULATE block would have in ARGA. This rule moves the argument values to the left one place.

Rule 51- the segment NEWLINE1 creates an OUTPUT segment in rule 89 and causes the output to start in column 1 of a new line each time it occurs. Any single symbol such as * that is separated by one or more spaces from the rest of the segments on the right side of a rule is printed out literally. The same action occurs for various rules in the GPSS morphology. Segment COLUMN7 tabs the output to column 7. Segment COMMENT becomes a copy of GSTMNT.

Rule 52- the PHRASE rule is part of the English encoding rules and causes a printout of the value of attribute CHARS of comment.

Rule 53- utilizes another rule from the English encoding rules to output the comment prior to the group of GPSS blocks representing each action in the IPD.

Rule 54- all the rest of the segments are processed by this rule which creates segments with attribute-value pairs representing the various fields of a GPSS statement. Segments NEWLINE2, COLUMN8, COLUMN13, and COLUMN19 are editing segments. They cause a new line to be begun, start the label field in column 2, the operation field in column 8, the mod field in column 13 and argument field in column 19. The attribute APTR is assigned the value of each argument attribute in turn. The attribute COMCNT of MEMORY and NCOM are used to determine when commas need to be output for missing arguments.

b. ARGDH, LABFLD, MODFLD, and ARGFLD

(Rules 55, 57, 59, 61)

Rule 55- processes segments that have attributes ARGD through ARGH.

Rule 57 and 59- creates ARG segment with a DATA attribute equal to the LABL or MOD attribute of their segment.

Rule 61- creates segments COMMAS and ARG with attributes NCOM and DATA, respectively.

c. XYOUT and LINE (Rules 63, 64, 65)

The object of these rules is to continue the processing required to output function X-Y pairs. The computed value of the attribute LAST of LINE limits the output to 4 pairs per line of output.

Rule 64- attribute DATA of ARG is set equal to a copy of the named record 'DECREC' which is given an attribute of NUM, the value of which is numerical or a record representing one of the coordinates. This is done when LINE is a segment which is a copy of EXPONREC or NORMREC, and also to specify that the exponential or normal distribution coordinates are to be output as decimals, as 'DECREC' has a SUP of 'DECIMAL'. If the indicator EXPNORM is not set, the attribute DATA is then placed equal to the NUM of DATA. When four pairs of coordinates have been printed out, i.e., the condition LC is greater than LAST exists, rule 65 applies and starts the sequence over by creating an XYOUT segment. The sequence ends when XYLAST of LINE is exceeded.

3. Morphology for encoding GPSS

a. OPFLD (Rules 67, 68)

Processing the NAME segment causes a printed output of the value of its CHARS attribute. As the value of the NAME attribute of the named records 'TERMINAT' and 'FVARIABLE' can only be 8 characters and the two words are 9 characters, their operation fields must be put out this way.

b. ARG (Rules 69, 70, 71)

The FORTRAN routine GETYPE determines the TYPE of cell and the ARG rules create a PARG segment, which would be a record, for further processing, or a number segment that causes numerical printout, or a NAME segment which causes a printout of characters.

c. PARG (Rules 72 through 85)

Processing of segments by these rules cause an output of arguments A through H of the GPSS block for various conditions on the left.

d. COMMAS (Rule 86)

Causes the output of a comma following each argument of the argument field except the last. If an argument is not required a comma is filled in for it.

e. NEWLINE1, NEWLINE2, NEWLINE8, COLUMN7, COLUMN8, COLUMN13, COLUMN19, NAME, NUMBER, and DECNUMB (Rules 88 through 98)

These are terminal rules that cause output editing, and output of integers, decimals and characters. Attribute 11 specifies how many lines to skip, attribute 12 specifies the column tab, attribute 13 the character output, attribute 14 integer output, and attribute 15 a decimal output.

4. Lexology for Encoding X-Vector

a. XSTMNT (Rules 100 through 104, 107, 109, 110, 118, 119, 120, 127)

Rule 100- the four conditions on the left cause XSTMNT to go to null. Segments with a SUP of 'COMMENT' or 'EQU' or 'SIMULATE' cause no output to the X-Vector.

Rules 101 and 102- facilities are treated as STORAGES with a capacity of 1 by the simulator- so segments with a SUP of 'SEIZE' or 'RELEASE' are converted to the STORAGE associated statement 'ENTER' or 'LEAVE'.

Rule 103- initiates output of eight random number seeds to the X-Vector. The value of the attribute FX of MEMORY, at any time, points to a specific X-Vector element one less than the beginning of free storage. The attribute INDX of MEMORY is deleted to insure that FX of MEMORY will be incremented by 1 each time a seed is loaded into the X-Vector. FX then takes the value of the last element of the allocation.

Rule 104- the two segments created here are processed further to produce one STORAGE and one QUEUE allocation. A QUEUE allocation is produced along with each STORAGE allocation.

Rule 107- this rule creates segments that produce a TABLE allocation. Segment SETDIR is created by this rule and in rules 105, 106, 109, 118, and 127 to store pointers to allocations of each entity. Later these pointers will be inserted in a directory for each entity at the end of all STORAGE, QUEUE, TABLE FUNCTION, VARIABLE, and BLOCK allocations. At that time a pointer to the directory will be inserted into its proper location in one of the first 32 elements. The attribute DIR of SETDIR points to a list of pointers for each entity, in this case 'TABDIR'. When the segment SETDIR is processed in rules 135 and 136, the value of the attribute LABL is incremented by 10 to give the attribute number of 'TABDIR' at which the allocation pointer is stored. The attribute LR of 'TABDIR' holds the value of the highest numbered attribute in the record for la

use in transferring the pointers to their directory in the X-Vector. When all allocations are made, there may be unassigned attributes in the list. An element is saved in the directory for the "missing" allocations as an allocation pointer's position in the directory provides an index to be used by the simulator to associate that entity with the various executable BLOCK arguments. This discussion of allocation pointer assignment was provided here to insure the continuity of the explanation rather than have it occur piecemeal in discussions of various rules. The values of attributes INDX and RIGHT of the X-Vector segments are the relative location in the allocation and the value to be placed in the right half of the element. The value of INDX is later added to the value of attribute FX of MEMORY to produce an absolute X-Vector index in which to store the value of RIGHT. FX of MEMORY acts as a base address for each allocation. INDX will never have the value 0. Element 6 contains the width of table frequency classes, element 8 the number of frequency classes in the table, and element 9 the upper limit of the lowest interval. The simulator uses the other elements in the allocation to collect statistics during the simulation. The above description of X-Vector attributes apply also to rules 105, 106, and to the segment XCODE in rule 109. The value of INDX of NXTALPTR is the relative location of the last element in the allocation. In this case, the value of ARGD was added to allow one element for each frequency class.

Rule 108- processes the NXTALPTR segment and places the sum of the base address, the value of attribute

FX of MEMORY, and the relative address, the value of INDX, in FX of MEMORY. This rule processes segments from rules 105, 106, 107.

Rule 109- segments are created that produce the first 5 elements of a FUNCTION allocation. The general explanation in rule 107 applies here. Elements 2 and 4 of this allocation require the codes -23 and -25 in the left half of elements 2 and 4. The 2 XCODE segments are created with attributes INDX and LEFT to cause entry of the above codes in elements 2 and 4. XCODE rules later make the value of the LEFT attribute negative. Therefore, the initial assignment is always a positive number. XARG segments are processed like ARG segments in rules 69 and 70. At later stages of processing, the value of the attribute INDX in rule 107 is placed into INDX of MEMORY to be added to the value of FX of MEMORY. Here, INDX of MEMORY is directly assigned a value.

Rule 110- the VALTYPE segment causes further processing to put the values 0 or 1 in the left and right halves of element 6 of a function allocation to signify whether the X and Y coordinates are integers or decimals. Values of the XXYOUT attributes are used in the processing of each attribute of the segment to extract X and Y values. The value of NOPTS is the number of pairs in the FUNCTION. The value of XALL, in this instance, is the relative location of each X value in the allocation and, added to the value of NOPTS, provides a relative location for each Y value.

Rule 118- segments that produce a VARIABLE l-location are created here. SETDIR is as described in r le

107. In the output of this and BLOCK allocations, rule 127, FX of MEMORY is incremented by 1 to locate absolutely each element in the allocation. The value of attribute INCREMENT is added to FX of MEMORY to locate the first allocation element. Subsequent incrementing is done in a loop. So that the value of INDX of MEMORY will not give erroneous locations, the attribute is erased.

Rule 119- these segments process the START statement. For the X-Vector, a mode code of 1 is put into element 1. The FORTRAN simulation routine takes certain initializing actions when a 1 is detected here. The code of 1 represents the action SIMULATE/START as opposed to RESET/START or CLEAR/START or just START. A more complete description is provided in Ref. 9. XVECTOR places the value of argument A of a START block in X-Vector element 2. The value represents the number of time elements that the simulation routine is to run. It is presently 1.

Rule 120- when all allocations are completed, this rule creates segments for all entities that are processed to build the entity directories and insert the proper values into elements 1 through 32. TRANSPTR causes the index of the last element assigned a value to be put into element 25 of the X-Vector. It locates the remaining free storage of the X-Vector, in which the simulation routine will build transaction allocations for the mobile entities that move through the model.

Rule 127- this rule processes all segments representing executable blocks, plus one that is not. SE DIR

is as described for rule 107. A test on the LABL attribute is done to determine whether the segment being processed is one representing the first block of a group that represents an action. The BLOKNO attribute is added to the original IPD action record for future reference. The value of BLOKNO will later be loaded into an element representing a block argument that specifies to which action a transfer must occur. The XVECTOR segment causes a code identifying the block to be loaded into the left half of the allocation 1st element and a pointer to the original action record into the right half.

b. XARGAC, XARGDH, XMODFLD, and XARGFLD

(Rules 128, 129, 131, 132, 133)

Rules 128 and 129- XARGFLD segments are treated in a manner similar to the ARGFLD segments of rules 54 and 55. The value of attribute APTR is the value of the ARGA through ARGH attributes assigned in the semology. The value of NCOM is used to skip allocation elements if an argument is not present, similar in manner to the output of commas for arguments not present in a GPSS block. Segments with a SUP or 'RMULT', 'SELECT', or 'TABLE' have more than three arguments so the XARGDH rule takes care of ARGD through ARGH attributes. In rule 128 the segment XMODFLD is created to fill the last element of any block allocation with a code of -33 in the left half and a code in the right half depending on the value of the MOD attribute assigned in the semology.

Rule 131- when the value of the last seed i placed in element 10 FX of MEMORY points to the free storage

available for entity allocations, normally a storage allocation.

Rule 132- the value of RIGHT depends on the value of the MOD attribute. If there is no MOD attribute, RIGHT then equals the value of attribute XMOD which comes only from segments with a SUP of 'GENERATE' or 'ASSIGN' in rule 31. A sup of 'GENERATE' places a pointer to a mobile entity in the right half. A sup of 'ASSIGN' places a code in the right half that signifies the operation to be accomplished between its argument values. At present, the only code is a 2, which signifies replacement.

c. STORAL and QALL (Rules 104, 105)

The explanation in rule 107 applies to these rules also. Element 2 of a STORAGE allocation is the number of available units, and element 10 holds a pointer to the storage IPD record in the right half. Element 8 of a QUEUE allocation holds a pointer to the STORAGE IPD for which the QUEUE is established.

d. XXYOUT, NXARG, VALTYPE, and YCHK

(Rules 111 through 116)

Rules 110 and 111- in order to handle the standard exponential and normal distributions, attribute DATA points to copy of the named record 'DECREC' with a SUP of 'DECIMAL'. The attribute NUM contains the value of the particular X coordinate. The value of attribute INDX of MEMORY is set to the relative location in the FUNCTION allocation. If the indicator EXPNORM is not set, then DM

points to the record that represents the FUNCTION. The NXARG rule does the same as above for the Y coordinate.

Rule 113- FX of MEMORY is set to the last element of the FUNCTION allocation. However, if the function is continuous, i.e., attribute DORC has a value of "C", additional elements must be added to the allocation for slope values. This is accomplished by adding the value of NOPTS to FX of memory.

Rules 113, 114, and 115- the explanation for rule 109 explains the use of these rules. All X-Y values of the standard exponential and normal distributions are considered to be decimal. The first coordinates of other segments representing functions are examined by checking the values of attributes 101 and 102 to determine whether they have a SUP of 'DECIMAL'.

e. PTRALL, PTRDIR, TRANSPTR, STUFBACK

(Rules 121, 122, 124, 125)

Rules 121 and 122- Rule 121 creates PTRALL to first set the pointer to the directory, then assigns the number of entities, and, finally, rule 122 sets up the directory.

Rule 124- sets the pointer to the final free storage, and sets an arbitrary number of parameters in element 24.

Rule 125- the value of the previously set attribute BLOKNO of the original IPD action record is placed in the proper X-Vector element.

5. Morphology for Encoding X-Vector.

a. XARG (Rules 137, 138)

The basic principles used are the same as that of rules 69 and 70, but the segments created are XPARG or XVECTOR. Since no characters are put into the X-Vector, XARG with no condition on the left goes to null.

b. XPARG (Rules 140 through 152)

Rule 140- the segments processed by this rule come from attribute ARGB of the STMNT segments created by the SUCSTMNT rules of the semology. ARGB represents the action a transaction will transfer to absolutely or on meeting some condition. However, that action record may not have been processed yet. In any event, there is not yet any way to store a block number in the X-Vector to which a transaction must transfer. This rule then does the next best thing. It stores a pointer to the original IPD action record in the X-Vector location that would normally hold a block number, and the value of the location of that element is stored as the value of an attribute in the named record 'BACKSTUF'. There may be a number of X-Vector elements with pointers to the same action record, as a transaction could transfer to a group of blocks representing an action from a number of other actions.

Rule 141- an entity's identification number is stored in an element of a block allocation.

Rule 142- processes a segment with a SUP of 'DECIMAL' to cause a decimal output to an X-Vector element. The first 3 segments created, i.e., OUTPUT, NUMBER, and

DECNUMB are created to output at the terminal the decimal value and the X-Vector location it is going to for checking purposes. The NULL segment stores the value of the X-Vector element in the attribute IX. The value of XV is the decimal value to be stored. A call to routine PUTFPX stores the value of XV at X-Vector location IX.

Rule 143- a segment in the set 'ABSTIME' requires a check to see if its units and the units of attribute TIMUNIT of MEMORY are the same. Creation of TIMARG causes further processing to do this. As the TIMARG rules are used jointly by the X-Vector and GPSS morphological rules, the indicator XVSW is set here to indicate to the TIMARG rules that an X-Vector segment is to be created to cause an output vice a number segment for the GPSS rules.

Rule 144- a segment in the set 'QUANVAL' causes an output of an integer.

Rule 145- a segment with a SUP of 'SNAREF' and a NUM attribute comes from rule 44 and causes an output of 22, the SNA code for FN,and the FN number.

Rule 146- this rule processes a segment record set in ARGA of rule 37 and causes an SNA code output of 12(Q) or 13(QA).

Rule 147- this segment originates in ARGB of rule 35. An output of an SNA code of 1(P) and the P number occurs.

Rule 148- the segment processed originates in attribute ARGA of rule 35 and causes an output of SNA code 3(MP) and a value of 1.

Rule 149- the segment processed originates in attribute ARGA of rule 22 and causes an output of SNA code 26(RN) and a number to specify the specific seed.

Rule 150- processing a segment with a SUP of 'NORMAL' causes an output of SNA code 24(V) and the V number.

Rule 151- the segment processed here originates at attribute ARGA of rule 26.

Rule 152- processing of a segment with an XYLAST attribute causes an output of SNA code 22(FN) along with the FN number.

c. XCODE and X-Vector (Rules 153, 154, 155, 156)

Rule 153- the simulation routine requires that all integer codes placed in the left half of an X-Vector element, except a pointer to an IPD be negative. This rule makes the value of attribute LEFT negative.

Rule 154- the value of the attribute INDX is the relative location of an element within an allocation. Only the value of attribute INDX of MEMORY is used in rule 154. The value of INDX is passed to INDX of MEMORY and the attribute INDX is erased.

Rule 155- If XVECTOR has no attribute IX, one is added and its value is the sum of FX of MEMORY and INDX of MEMORY. If there is no attribute INDX of MEMORY then FX of MEMORY is incremented by one and FX acts as a pointer to an element, not an allocation base address. This technique is used in outputting VARIABLE and BLOCK allocations.

Rule 156- the first four segments, i.e., OUTPUT, NUMBER, NUMBER and NUMBER are presently created to output at the terminal, on a new line, the values placed in the left and right half of an X-Vector element and the X-Vector element number for checking purposes. This line of output comes out just below the equivalent GPSS statement output. Actions taken at the creation of the NULL segment are similar to those of rule 142 except that values are placed into the left and right halves of an X-Vector element. Either or both values may be zero. The routines PUTLX and PUTRX are called to accomplish the action.

d. SNAS (Rules 157, 158)

Rule 157- each argument in a BLOCK allocation occupies a unique position in the allocation. Therefore, even if the argument is not filled, the allocation must assign an element. The values of attributes NSNA and SNACNT of MEMORY are used in conjunction with each other to increment the value of FX of MEMORY by 1 for each missing argument within a group of arguments. No elements are assigned for missing arguments at the end of the allocation.

e. TIMARG and TNUMBER (Rules 159 through 164)

Rules 159 through 164- this set of rules is used by both the BPSS and X-Vector morphological rules to adjust the value of problem time parameters to conform to the problem basic time unit. For instance if the basic time unit were 30 seconds and the problem were to run for 60 minutes a figure of 120 basic time units would be used in the timing loop. In rules 159 and 160 an XVSW attribute is created if the XV N

indicator is set in the segment being processed. Otherwise, processing is the same in rules 159 through 162. Once the time figure has been adjusted the TNUMBER rules create X-Vector or NUMBER segments depending on whether attribute XVSW is set.

V. CONCLUSIONS AND RECOMMENDATIONS

The objective of this thesis stated in the Introduction was met. The link between the simulation routine and its source data, the IPD, is now complete and opens a whole new realm of further development possibilities. In integrating the GPSS and X-Vector rules, greater care had to be taken to conform to the three stratificational levels described by Lamb. The resulting rule module had a common semology, but separate lexologies and morphologies.

Further development of NLPQ lies in exploiting the fact that the link has been made between the IPD and the simulation routine. Future research should concentrate on how best to conduct the communication between the two in an interactive mode to perform simulation analyses "On-Line". This would necessarily involve a determination of the following items:

1. What questions would be asked about the problem?
2. What output should be expected from the simulation?
3. What form would it take, tabular or English text?
4. How may the clear, reset, or restart capabilities of a GPSS program be implemented?

5. How can the program be rerun with changed parameters?
6. How are the results of various simulations to be compared?

The primary goal should be to determine the best way in which to perform queuing simulations in a fully interactive mode at the terminal, developing techniques and methods that could be applied to a more general system.

APPENDIX A

LISTING OF ROUTINES

```
C                                                          GETYPE
 3100 LOCA9 = LOC(A9,SEGMNT)
      CEL = CELL(LOCA9)
      ADDR = TYPE
      TYPE = 0
      CELL(LOCA9) = CEL
      GO TO 1200
      INTEGER*2 XHW(4)
      INTEGER*4 XFW(2),X(300)
      REAL*4 XFFW(2)
      REAL*8 XDW
      EQUIVALENCE  (XDW,XFFW,XFW,XHW)
      DATA AIX/8/,AVX/9/

C                                                          PUTX
 3110 X(HVAL(AIX,SEGMNT)) = HVAL(AVX,SEGMNT)
      GO TO 1200

C                                                          PUTLX
 3120 IX = HVAL(AIX,SEGMNT)
      XFW(1) = X(IX)
      XHW(1) = HVAL(AVX,SEGMNT)
      X(IX) = XFW(1)
      GO TO 1200

C                                                          PUTRX
 3130 IX = HVAL(AIX,SEGMNT)
      XFW(1) = X(IX)
      XHW(2) = HVAL(AVX,SEGMNT)
      X(IX) = XFW(1)
      GO TO 1200

C                                                          PUTFPX
 3140 XFFW(1) = 0.001*HVAL(AVX,SEGMNT)
      X(HVAL(AIX,SEGMNT)) = XFW(1)
      GO TO 1200

C                                                          GETX
 3150 XFW(1) = X(HVAL(AIX,SEGMNT))
      CALL  HSTORE(XHW(2),AVX,SEGMNT)
      GO TO 1200

C                                                          GETLX
 3160 XFW(1) = X(HVAL(AIX,SEGMNT))
      CALL HSTORE(XHW(1),AVX,SEGMNT)
      GO TO 1200

C                                                          GETRX
 3170 GO TO 3150


C                                                          GETFPX
 3180 XFW(1) = X(HVAL(AIX,SEGMNT))
      VX = 1000.0*XFFW(1)
      CALL HSTORE(VX,AVX,SEGMNT)
      GO TO 1200

C                                                          ZEROXV
 3190 DO 3195 IX = 1,300
 3195 X(IX) = 0
      GO TO 1200
```

APPENDIX B

ANOTHER SAMPLE PROBLEM(THE HARBOR PROBLEM)

NAMED RECORDS DEFINING THE HARBOR PROBLEM IPD.

```
NAMED RECORDS:
'REC1'     ('ARRIV',IDNO=1,SUCC='REC2',AGENT='REC11',
             LOCATION='REC71',IETM='REC41',ASNDISTR='REC47')
'REC2'     ('UNLOAD',IDNO=2,PRED='REC1',SUCC='REC61',
             AGENT='REC11',GOAL='REC13',LOCATION='REC72',
             DURATION='REC42')
'REC3'     ('UNLOAD',IDNO=3,PRED='REC2',SUCC='REC62',
             AGENT='REC15',GOAL='REC13',LOCATION='REC74',
             DURATION='REC44')
'REC4'     ('UNLOAD',IDNO=4,PRED='REC2',SUCC='REC62',
             AGENT='REC17',GOAL='REC13',LOCATION='REC75',
             DURATION='REC45')
'REC5'     ('UNLOAD',IDNO=5,PRED='REC2',SUCC='REC62',
             AGENT='REC19',GOAL='REC13',LOCATION='REC76',
             DURATION='REC46')
'REC6'     ('LOAD',IDNO=6,PRED='REC96',SUCC='REC7',
             AGENT='REC11',GOAL='REC13',LOCATION='REC72',
             DURATION='REC210')
'REC7'     ('WAIT',IDNO=7,PRED='REC97',SUCC='REC8',
             AGENT='REC11',LOCATION='REC77',DURATION='REC81')
'REC8'     ('LEAV',IDNO=8,PRED='REC7',AGENT='REC11',
             LOCATION='REC71')

'REC11'    ('SHIP',IDNO=1,CONSUMP='REC212', IDNAME="SHIP",
                  CLASATR='COLOR')
'REC12'    ('PORT',IDNO=2,QUANTITY=1,STORIND=1, CAPACITY=
             'REC221',IDNAME="PORT")
'REC13'    ('CARGO',IDNO=3, IDNAME="CARGO")
'REC14'    ('DOCK',IDNO=4,LOCATION='REC73',QUANTITY=3,
             CAPACITY='REC212',STORIND=1, IDNAME="DOCKS")
'REC15'    ('SHIP',IDNO=5,QUANTITY='REC205',COLOR='GREEN',
             STRUC='REC11', IDNAME="GSHIP")
'REC16'    ('PIER',IDNO=6,LOCATION='REC73',QUANTITY=2,
             CAPACITY='REC212',STORIND=1, IDNAME="PIERS")
'REC17'    ('SHIP',IDNO=7,QUANTITY='REC207',COLOR='RED',
             STRUC='REC11', IDNAME="RSHIP")
'REC18'    ('DEPOT',IDNO=8,LOCATION='REC73',QUANTITY=1,
             CAPACITY='REC220',STORIND=1, IDNAME="DEPOT")
'REC19'    ('SHIP',IDNO=9,QUANTITY='REC208',COLOR='BLUE',
             STRUC='REC11', IDNAME="BSHIP",INTWIDTH=50,
             NUMINT=22,LOWINT=200)
'REC20'    ('BARGE',IDNO=10,LOCATION='REC73',QUANTITY=1,
             CAPACITY='REC212', IDNAME="BARGE")
'REC21'    ('HARBOR',IDNO=11,LOCATION='REC73',QUANTITY=1,
             STORIND=1, CAPACITY='REC221', IDNAME="HRBR")

'REC41'    ('UNIFORM',MEAN='REC201',RANGE='REC202')
'REC42'    ('NORMAL',MEAN='REC43',STDEV='REC48')
'REC43'    ('TYPTABL',FNARG='REC203',DORC="D",
             XYLAST=106,@101='REC15',@102='REC213',
             @103='REC17',@104='REC214',@105='REC19',
             @106='REC201')
'REC44'    ('NORMAL',MEAN='REC202',STDEV='REC206')
'REC45'    ('EXPON',MEAN='REC202')
'REC46'    ('EXPON',MEAN='REC209')
'REC47'    ('TYPDIST',PNUM='REC212',FNARG='REC211',
             DORC="D",XYLAST=106,@101='REC215',@102='REC15',
             @103='REC216',@104='REC17',@105='REC217',
             @106='REC19')
'REC48'    ('TYPTABL',FNARG='REC203',DORC="D",
             XYLAST=106,@101='REC15',@102='REC218',
             @103='REC17',@104='REC202',@105='REC19',
             @106='REC209')
```

```
'REC61'    ('PTYP',SUCARG='REC203',XYLAST=106,@101='REC15',
             @102='REC3',@103='REC17',@104='REC4',
             @105='REC19',@106='REC5',FNARG='REC203',DORC="D"
'REC62'    ('FRACTNL',SUCARG='REC211',XYLAST=104,
             @101='REC219',@102='REC6',@103='REC217',
             @104='REC7',FNARG='REC211',DORC="D")

'REC71'    ('AT',LOCOBJ='REC12')
'REC72'    ('AT',LOCOBJ='REC14')
'REC73'    ('IN',LOCOBJ='REC12')
'REC74'    ('AT',LOCOBJ='REC16')
'REC75'    ('AT',LOCOBJ='REC18')
'REC76'    ('AT',LOCOBJ='REC20')
'REC77'    ('IN',LOCOBJ='REC21')

'REC81'    ('COND1',CONDENTY='REC20')

MOBLIST    ('RECLIST',LASTREC=14,@11='REC11',@12='REC15',
             @13='REC17',@14='REC19')
STALIST    ('RECLIST',LASTREC=17,@11='REC12',@12='REC13',
             @13='REC14',@14='REC16',@15='REC18',
             @16='REC20',@17='REC21')
ACTNLIST   ('RECLIST',LASTREC=18,@11='REC1',@12='REC2',
             @13='REC3',@14='REC4',@15='REC5',@16='REC6',
             @17='REC7',@18='REC8')
DSTRLIST   ('RECLIST',LASTREC=18,@11='REC41',@12='REC42',
             @13='REC43',@14='REC44',@15='REC45',@16='REC46',
             @17='REC47',@18='REC48')
SCSRLIST   ('RECLIST',LASTREC=15,@11='REC2',@12='REC61',
             @13='REC62',@14='REC7',@15='REC8')

'REC96'    ('PREDLIST',LASTREC=13,@11='REC3',@12='REC4',
             @13='REC5')
'REC97'    ('PREDLIST',LASTREC=14,@11='REC3',@12='REC4',
             @13='REC5',@14='REC6')

'REC201'   ('HOUR',NUM=5)
'REC202'   ('HOUR',NUM=1)
'REC203'   ('PARAMNO',NUM=1)
'REC204'   ('DAY',NUM=4)
'REC205'   ('PERCENT',NUM=20)
'REC206'   ('MINUTE',NUM=15)

'REC207'   ('PERCENT',NUM=30)
'REC208'   ('PERCENT',NUM=50)
'REC209'   ('MINUTE',NUM=90)
'REC210'   ('HOUR',NUM=2)
'REC211'   ('RANDM')
'REC212'   ('UNIT',NUM=1)
'REC213'   ('HOUR',NUM=3)
'REC214'   ('HOUR',NUM=4)
'REC215'   ('DECIMAL',NUM=200)
'REC216'   ('DECIMAL',NUM=500)
'REC217'   ('DECIMAL',NUM=1000)
'REC218'   ('MINUTE',NUM=30)
'REC219'   ('DECIMAL',NUM=400)
'REC220'   ('UNIT',NUM=4)
'REC221'   ('UNIT', NUM=1000)
'REC222'   ('MINUTE',NUM=1)

SETMEM     (PROBTIME(MEM)='REC204',TIMUNIT(MEM)='REC222')

UCELLS:

:END OF FILE:
```

ENCODED ENGLISH PROBLEM DESCRIPTION.

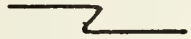
THE SHIPS ARRIVE AT THE PORT. THE TIME BETWEEN ARRIVALS OF THE SHIPS AT THE PORT IS UNIFORMLY DISTRIBUTED, WITH A MEAN OF 5 HOURS AND A HALF-RANGE OF 1 HOUR. 20 PERCENT OF THE SHIPS ARE GREEN, 30 PERCENT ARE RED, AND THE REST ARE BLUE. AFTER ARRIVING AT THE PORT, THE SHIPS UNLOAD CARGO AT A DOCK IN THE PORT. THERE ARE 3 DOCKS IN THE PORT. THE TIME FOR THE SHIPS TO UNLOAD CARGO AT A DOCK IN THE PORT IS NORMALLY DISTRIBUTED, WITH A MEAN OF 3 HOURS FOR THE GREEN SHIPS, 4 HOURS FOR THE RED SHIPS, AND 5 HOURS FOR THE BLUE SHIPS AND A STANDARD DEVIATION OF 30 MINUTES FOR THE GREEN SHIPS, 1 HOUR FOR THE RED SHIPS, AND 90 MINUTES FOR THE BLUE SHIPS. AFTER UNLOADING CARGO AT A DOCK IN THE PORT, THE GREEN SHIPS UNLOAD CARGO AT A PIER IN THE PORT, THE RED SHIPS UNLOAD CARGO AT THE DEPOT IN THE PORT, AND THE BLUE SHIPS UNLOAD CARGO AT THE BARGE IN THE PORT. THERE ARE 2 PIERS IN THE PORT. THE TIME FOR THE GREEN SHIPS TO UNLOAD CARGO AT A PIER IN THE PORT IS NORMALLY DISTRIBUTED, WITH A MEAN OF 1 HOUR AND A STANDARD DEVIATION OF 15 MINUTES. AFTER UNLOADING CARGO AT A PIER IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT A DOCK IN THE PORT FOR 2 HOURS, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. THE CAPACITY OF THE DEPOT IS 4 SHIPS. THE TIME FOR THE RED SHIPS TO UNLOAD CARGO AT THE DEPOT IN THE PORT IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 1 HOUR. AFTER UNLOADING CARGO AT THE DEPOT IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT A DOCK IN THE PORT FOR 2 HOURS, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. THE TIME FOR THE BLUE SHIPS TO UNLOAD CARGO AT THE BARGE IN THE PORT IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 90 MINUTES. AFTER UNLOADING CARGO AT THE BARGE IN THE PORT, 40 PERCENT OF THE SHIPS LOAD CARGO AT A DOCK IN THE PORT FOR 2 HOURS, AND THE REST WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. AFTER LOADING CARGO AT A DOCK IN THE PORT, THE SHIPS WAIT IN THE HARBOR IN THE PORT UNTIL THE BARGE IS AVAILABLE. AFTER WAITING IN THE HARBOR IN THE PORT, THE SHIPS LEAVE THE PORT.

THE SIMULATION IS TO BE RUN FOR 4 DAYS, USING A BASIC TIME UNIT OF 1 MINUTE.

GPSS/X-VECTOR OUTPUT.

```
        SIMULATE
        RMULT          277,423,815,121,655,531,999,813
    3   0   277
    4   0   423
    5   0   815
    6   0   121
    7   0   655
    8   0   531
    9   0   999
   10   0   813
PORT    EQU            2,S,Q
2       STORAGE        1000
   34   0   1000
   42   0   6397
   50   6397   0
CARGO   EQU            3,T
   52   0   1
   60   0   4929
   68   4929   0
DOCKS   EQU            4,S,Q
4       STORAGE        3
   70   0   3
   78   0   3642
   86   3642   0
PIERS   EQU            6,S,Q
```

```
6       STORAGE     2
    88   0  2
    96   0  3672
    104  3672  0
DEPOT EQU           8,S,Q
8       STORAGE     4
    106  0  4
    114  0  6979
    122  6979  0
BARGE EQU           10,F,Q
    124  0  1
    132  0  4082
    140  4082  0
HRBR  EQU           11,S,Q
11      STORAGE     1000
    142  0  1000
    150  0  5130
    158  5130  0
GSHIP EQU           5,T
5       TABLE       M1,1,1,2
    164  0  1
    166  0  2
    167  0  1
RSHIP EQU           7,T
7       TABLE       M1,1,1,2
    175  0  1
    177  0  2
    178  0  1
BSHIP EQU           9,T
9       TABLE       M1,200,50,22
    186  0  50
    188  0  22
    189  0  200
1       FUNCTION    RN1,C24
    212  -26  1
    213  -23  6141
    215  -25  0
    216  1  24
0.0,0.0/.100,.104/.200,.222/.300,.355/
.400,.509/.500,.690/.600,.915/.700,1.200/
.750,1.390/.800,1.600/.840,1.830/.882,2.120/
.900,2.300/.920,2.520/.940,2.810/.950,2.990/
.960,3.200/.970,3.500/.980,3.900/.990,4.600/
.995,5.300/.998,6.200/.999,7.000/1.000,8.000/
    217  1  1
    218  0.0
    242  0.0
    219  .100
    243  .104
```



```
    240  .999
    264  7.000
    241  1.000
    265  8.000
2       FUNCTION    RN2,C29
    290  -26  2
    291  -23  4421
    293  -25  0
    294  1  29
0.0,-3.000/.012,-2.250/.027,-1.930/.043,-1.720/
.062,-1.540/.084,-1.380/.104,-1.260/.131,-1.120/
.158,-1.000/.187,-.890/.230,-.740/.267,-.620/
.334,-.430/.432,-.170/.500,0.0/.568,.170/
.666,.430/.732,.620/.770,.740/.813,.890/
.841,1.000/.869,1.120/.896,1.260/.916,1.380/
.938,1.540/.957,1.720/.973,1.930/.988,2.250/
1.000,3.000/
    295  1  1
    296  0.0
    325  -3.000
```

```
     297  .012
     326  -2.250
     ——————2——————
     323  .988
     352  2.250
     324  1.000
     353  3.000
 3       FUNCTION    P1,D3
     383  -1  1
     384  -23  5569
     386  -25  0
     387  2  3
GSHIP,180/RSHIP,240/BSHIP,300/
     388  0  0
     389  0  5
     392  0  180
     390  0  7
     393  0  240
     391  0  9
     394  0  300
 4       FUNCTION    RN3,D3
     395  -26  3
     396  -23  3837
     398  -25  0
     399  2  3
.200,GSHIP/.500,RSHIP/1.000,BSHIP/
     400  1  0
     401  .200
     404  0  5
     402  .500
     405  0  7
     403  1.000
     406  0  9
 5       FUNCTION    P1,D3
     407  -1  1
     408  -23  6043
     410  -25  0
     411  2  3
GSHIP,30/RSHIP,60/BSHIP,90/
     412  0  0
     413  0  5
     416  0  30
     414  0  7
     417  0  60
     415  0  9
     418  0  90
 6       FUNCTION    P1,D3
     419  -1  1
     420  -23  4866
     422  -25  0
     423  2  3
GSHIP,ACT3/RSHIP,ACT4/BSHIP,ACT5/
     424  0  0
     425  0  5
     428  0  4010
     426  0  7
     429  0  6576
     427  0  9
     430  0  5080
 7       FUNCTION    RN4,D2
     431  -26  4
     432  -23  6582
     434  -25  0
     435  2  2
.400,ACT6/1.000,ACT7/
     436  1  0
     437  .400
     439  0  5736
     438  1.000
     440  0  7067
```

```
 1      FVARIABLE  FN3+FN5*FN2
    441   -22   3
    442   -22   5
    443   -22   2
    444   -30   3
    445   -30   1
    446   -25   0
 2      FVARIABLE  60+15*FN2
    447   0   60
    448   0   15
    449   -22   2
    450   -30   3
    451   -30   1
    452   -25   0
*
*       THE SHIPS ARRIVE AT THE PORT.
        GENERATE   300,60
    453   1   3901
    454   0   300
    455   0   60
    456   -33   6155
        ASSIGN     1,FN4
    457   4   3901
    458   0   1
    459   -22   4
    460   -33   2
        ENTER      PORT
    461   6   3901
    462   0   2
    463   -33   0
*
*       THE SHIPS UNLOAD CARGO AT A DOCK.
 ACT2   QUEUE      DOCKS
    464   8   3374
    465   0   4
    466   -33   0
        ENTER      DOCKS
    467   6   3374
    468   0   4
    469   -33   0
        DEPART     DOCKS
    470   9   3374
    471   0   4
    472   -33   0
        ADVANCE    V1
    473   2   3374
    474   -24   1
    475   -33   0
        LEAVE      DOCKS
    476   7   3374
    477   0   4
    478   -33   0
        TRANSFER   ,FN6
    479   3   3374
    481   -22   6
    482   -33   0
*
*       THE GREEN SHIPS UNLOAD CARGO AT A PIER.
 ACT3   QUEUE      PIERS
    483   8   4010
    484   0   6
    485   -33   0
        ENTER      PIERS
    486   6   4010
    487   0   6
    488   -33   0
        DEPART     PIERS
    489   9   4010
    490   0   6
    491   -33   0
        ADVANCE    V2
    492   2   4010
```

```
     493   -24   2
     494   -33   0
        LEAVE        PIERS
     495   7   4010
     496   0   6
     497   -33   0
        TRANSFER     ,FN7
     498   3   4010
     500   -22   7
     501   -33   0
*
*       THE RED SHIPS UNLOAD CARGO AT THE DEPOT.
 ACT4   QUEUE        DEPOT
     502   8   6576
     503   0   8
     504   -33   0
        ENTER        DEPOT
     505   6   6576
     506   0   8
     507   -33   0
        DEPART       DEPOT
     508   9   6576
     509   0   8
     510   -33   0
        ADVANCE      60,FN1
     511   2   6576
     512   0   60
     513   -22   1
     514   -33   0
        LEAVE        DEPOT
     515   7   6576
     516   0   8
     517   -33   0
        TRANSFER     ,FN7
     518   3   6576
     520   -22   7
     521   -33   0
*
*       THE BLUE SHIPS UNLOAD CARGO AT THE BARGE.
 ACT5   QUEUE        BARGE
     522   8   5080
     523   0   10
     524   -33   0
        SEIZE        BARGE
     525   6   5080
     526   0   10
     527   -33   0
        DEPART       BARGE
     528   9   5080
     529   0   10
     530   -33   0
        ADVANCE      90,FN1
     531   2   5080
     532   0   90
     533   -22   1
     534   -33   0
        RELEASE      BARGE
     535   7   5080
     536   0   10
     537   -33   0
        TRANSFER     ,FN7
     538   3   5080
     540   -22   7
     541   -33   0
*
*       THE SHIPS LOAD CARGO AT A DOCK.
 ACT6   QUEUE        DOCKS
     542   8   5736
     543   0   4
     544   -33   0
        ENTER        DOCKS
     545   6   5736
```

```
     546   0   4
     547  -33   0
        DEPART      DOCKS
     548   9   5736
     549   0   4
     550  -33   0
        ADVANCE     120
     551   2   5736
     552   0   120
     553  -33   0
        LEAVE       DOCKS
     554   7   5736
     555   0   4
     556  -33   0
*
*       THE SHIPS WAIT IN THE HARBOR.
 ACT7   QUEUE       HRBR
     557   8   7067
     558   0   11
     559  -33   0
        ENTER       HRBR
     560   6   7067
     561   0   11
     562  -33   0
        DEPART      HRBR
     563   9   7067
     564   0   11
     565  -33   0
        GATE NU     BARGE
     566   16  7067
     567   0   10
     568  -33   3
        LEAVE       HRBR
     569   7   7067
     570   0   11
     571  -33   0
*
*       THE SHIPS LEAVE THE PORT.
 ACT8   TABULATE    P1
     572   12  4104
     573   -3  1
     574   -1  1
     575   0   -1
     576  -33   0
        LEAVE       PORT
     577   7   4104
     578   0   2
     579  -33   0
        TERMINATE
     580   10  4104
     581  -33   0
*
*       TIMING LOOP
        GENERATE    5760
     582   1   0
     583   0   5760
     584  -33   0
        TERMINATE   1
     585   10  0
     586   0   1
     587  -33   0
        START       1
    1   0   1
    2   0   1
        END
     13    0   587
     12    0   11
     588   0   0
     589   0   32
     590   0   50
     591   0   68
     592   0   0
```

```
593   0   86
594   0   0
595   0   104
596   0   0
597   0   122
598   0   140
15   0   598
14   0   11
599   0   0
600   0   42
601   0   60
602   0   78
603   0   0
604   0   96
605   0   0
606   0   114
607   0   0
608   0   132
609   0   150
21   0   609
20   0   9
610   0   0
611   0   0
612   0   0
613   0   0
614   0   158
615   0   0
616   0   169
617   0   0
618   0   180
17   0   618
16   0   7
619   0   211
620   0   289
621   0   382
622   0   394
623   0   406
624   0   418
625   0   430
32   0   625
31   0   2
626   0   440
627   0   446
19   0   627
18   0   42
628   0   452
629   0   456
630   0   460
631   0   463
632   0   466
633   0   469
634   0   472
635   0   475
636   0   478
637   0   482
638   0   485
639   0   488
640   0   491
641   0   494
642   0   497
643   0   501
644   0   504
645   0   507
646   0   510
647   0   514
648   0   517
649   0   521
650   0   524
651   0   527
652   0   530
653   0   534
654   0   537
```

```
655   0   537
656   0   544
657   0   547
658   0   550
659   0   553
660   0   556
661   0   559
662   0   562
663   0   565
664   0   568
665   0   571
666   0   576
667   0   579
668   0   581
669   0   584
24    0   3
25    0   669
428   0   10
429   0   16
430   0   22
439   0   28
440   0   33
```

APPENDIX C

THE GPSS/X-VECTOR RULES

```
*****************************************************************************/
INDICATORS:
          GPSSSW      11
          XVSW        12
          EXPNORM     13
          NOCOPY      14
*****************************************************************************/
ATTRIBUTES:
          IX   8, XV   9
          ARGA  11, ARGB  12, ARGC  13, ARGD  14, ARGE  15, ARGF  16,
          ARGG  17, ARGH  18
*****************************************************************************/
ROUTINES:
          GETYPE  10, PUTX  11, PUTLX  12, PUTRX  13, PUTFPX  14,
          GETX  15, GETLX  16, GETRX  17, GETFPX  18, ZEROXV  19
*****************************************************************************/
NAMED RECORDS:
  (1)     SEEDS     ('RMULT',@11=277,@12=423,@13=815,@14=121,@15=655,@16=531,
                     @17=999,@18=813)

  (2)     EXPONREC  (@101=0,@102=0,@103=100,@104=104,@105=200,@106=222,
                     @107=300,@108=355,@109=400,@110=509,@111=500,@112=690,
                     @113=600,@114=915,@115=700,@116=1200,@117=750,@118=1390,
                     @119=800,@120=1600,@121=840,@122=1830,@123=882,@124=2120,
                     @125=900,@126=2300,@127=920,@128=2520,@129=940,@130=2810,
                     @131=950,@132=2990,@133=960,@134=3200,@135=970,@136=3500,
                     @137=980,@138=3900,@139=990,@140=4600,@141=995,@142=5300,
                     @143=998,@144=6200,@145=999,@146=7000,@147=1000,@148=8000,
                     XYLAST=148,DORC="C",FNARG='RANDMREC',IDNO=1,EXPNORM)

  (3)     NORMREC   (@101=0,@102=0-3000,@103=12,@104=0-2250,@105=27,@106=0-1930,
                     @107=43,@108=0-1720,@109=62,@110=0-1540,@111=84,
                     @112=0-1380,@113=104,@114=0-1260,@115=131,@116=0-1120,
                     @117=158,@118=0-1000,@119=187,@120=0-890,@121=230,
                     @122=0-740,@123=267,@124=0-620,@125=334,@126=0-430,
                     @127=432,@128=0-170,@129=500,@130=0,@131=568,@132=170,
                     @133=666,@134=430,@135=732,@136=620,@137=770,@138=740,
                     @139=813,@140=890,@141=841,@142=1000,@143=869,@144=1120,
                     @145=896,@146=1260,@147=916,@148=1380,@149=938,@150=1540,
                     @151=957,@152=1720,@153=973,@154=1930,@155=988,@156=2250,
                     @157=1000,@158=3000,
                     XYLAST=158,DORC="C",FNARG='RANDMREC',IDNO=2,EXPNORM)
```

```
 (4)      GENERATE (BTCODE=1)
 (5)      ADVANCE  (BTCODE=2)
 (6)      TRANSFER (BTCODE=3)
 (7)      ASSIGN   (BTCODE=4)
 (8)      MARK     (BTCODE=5)
 (9)      ENTER    (BTCODE=6)
(10)      SEIZE    (BTCODE=6)
(11)      LEAVE    (BTCODE=7)
(12)      RELEASE  (BTCODE=7)
(13)      QUEUE    (BTCODE=8)
(14)      DEPART   (BTCODE=9)
(15)      TERMINATE(BTCODE=10)
(16)      TEST     (BTCODE=11)
(17)      TABULATE (BTCODE=12)
(18)      SAVEVALUE(BTCODE=13)
(19)      GATE     (BTCODE=16)
(20)      SELECT   (BTCODE=21)

(21)      STORDIR  (DIRPTR=13)
(22)      QDIR     (DIRPTR=15)
(23)      TABDIR   (DIRPTR=21)
(24)      FNDIR    (DIRPTR=17)
(25)      VARDIR   (DIRPTR=32)
(26)      BLKDIR   (DIRPTR=19)
(27)      SVDIR    (DIRPTR=23)

(28)      DECREC   ('DECIMAL')
(29)      SNAREC   ('SNAREF')
(30)      TRANSREC ('TRANSTIM')
(31)      COND1    ('COND',SMOD="SNF",FMOD="NU")
(32)      COND2    ('COND',SMOD="SF",FMOD="U")
(33)      BACKSTUF ('MISC')

(34)      SETMEM   (GPSSSW(MEM),XVSW(MEM))
*****************************************************************************/
SEMOLOGY FOR ENCODING GPSS/X-VECTOR:

 (1)      GPSSPROG(¬QUESTSW(MEM))   -->   INITIALGPSS
                   SETAGL(%ACPTR(MEM),LC=11)
                   SCSRREC(IDNO=MFNID(MEM),IDNO(ENTY)=IDNO,
                   STMNT('SIMULATE')   STMNT(%'SEEDS')
                   SELIST(%SEPTR(MEM),LC=11)
                   MELIST(%MEPTR(MEM),LC=11)    EXPFUNC    NORMFUNC
                   DISTLIST(%DISTPTR(MEM),LC=11)
                   SUCLIST(%SUCPTR(MEM),LC=11)
                   DSTLIST(%DISTPTR(MEM),LC=11)
                   ACLIST(%ACPTR(MEM),LC(MEM)=11)    TMLOOP
```

```
(2)     GPSSPROG   -->   NULL
(3)     INITIALGPSS    -->
                NULL(GRNNO(MEM)=0,XRNNO(MEM)=0,
                     LR('BACKSTUF')=10,
                     LR('STORDIR')=10,LR('QDIR')=10,
                     LR('TABDIR')=10,LR('FNDIR')=10,
                     LR('VARDIR')=10,LR('BLKDIR')=10,
                     LR('SVDIR')=10,XVSW(MEM).NE.0,ZEROXV)
*************************************************************************/
(4)     SELIST(LC.LE.LASTREC)    -->
                STENTITY(%@LC(SELIST)(SELIST),IPDP=@LC(SELIST)(SELIST))
                SELIST(LC=LC+1)
(5)     SELIST    -->    NULL
(6)     MELIST(LC.LE.LASTREC)    -->    MENTITY(%@LC(MELIST)(MELIST))
                MELIST(LC=LC+1)
(7)     MELIST    -->    NULL
(8)     STENTITY    -->
                EQUCARD(%STENTITY)
                STMNT('STORAGE',LABL=IDNO(STENTITY),
                     ARGA=QUANTITY($STRUC(STENTITY))
                     *NUM(CAPACITY($STRUC(STENTITY))),
                     IPDP(STENTITY))
(9)     MENTITY    -->    EQUCARD(%MENTITY)    TABLECARD(%MENTITY)
(10)    EQUCARD(IDNAME,¬CLASATR)    -->
                STMNT('EQU',LABL=IDNAME(EQUCARD),ARGA=IDNO(EQUCARD),
                     ARGB="S",ARGC="Q",¬STORIND($STRUC(EQUCARD)),ARGB="F",
                     EQUCARD¬$'STATENTY',ARGB="T",-ARGC)
(11)    EQUCARD --> NULL
(12)    TABLECARD(¬CLASATR)    -->
                STMNT('TABLE', LABL=IDNO(TABLECARD),
                     ARGA="M1",ARGB=LOWINT($STRUC(TABLECARD)),
                     ARGC=INTWIDTH($STRUC(TABLECARD)),
                     ARGD=NUMINT($STRUC(TABLECARD)))
(13)    TABLECARD --> NULL
*************************************************************************/
(14)    EXPFUNC(EXPUSED(MEM))    -->    FNDEF(%'EXPONREC',IPDP='EXPONREC')
(15)    EXPFUNC    -->    NULL
(16)    NORMFUNC(NORMUSED(MEM))    -->    FNDEF(%'NORMREC',IPDP='NORMREC')
(17)    NORMFUNC    -->    NULL
(18)    DISTLIST(LC.LE.LASTREC)    -->
                FNDEF(%@LC(DISTLIST)(DISTLIST),
                     IPDP=@LC(DISTLIST)(DISTLIST))
                DISTLIST(LC=LC+1)
(19)    DISTLIST    -->    NULL
(20)    SUCLIST(LC.LE.LASTREC)    -->
                FNDEF(%@LC(SUCLIST)(SUCLIST),IPDP=@LC(SUCLIST)(SUCLIST))
                SUCLIST(LC=LC+1)
(21)    SUCLIST    -->    NULL
```

```
(22)     FNDEF(XYLAST)   -->
                  STMNT('FUNCTION', LABL=IDNO(FNDEF),
                        ARGA=FNARG(FNDEF),TEMP=XYLAST(FNDEF)-100,
                        ARGB=%'ARGREC',FNTYPE(ARGB)=DORC(FNDEF),
                        NOPTS(ARGB)=TEMP/2,IPDP(FNDEF))
                  STMNT(%FNDEF,LC=101)
(23)     FNDEF   -->   NULL
*************************************************************************/
(24)     DSTLIST(LC.LE.LASTREC)   -->
                  DISTDEF(%@LC(DSTLIST)(DSTLIST))    DSTLIST(LC=LC+1)
(25)     DSTLIST   -->   NULL
(26)     DISTDEF('NORMAL')   -->
                  STMNT('FVARIABL',LABL=IDNO(DISTDEF),
                        ARGA=%'ARGREC',MEAN(ARGA)=MEAN(DISTDEF),
                        STDEV(ARGA)=STDEV(DISTDEF))
(27)     DISTDEF   -->   NULL
*************************************************************************/
(28)     ACLIST(LC(MEM).LE.LASTREC)   -->
                  ACTT(%@LC(MEM)(ACLIST),IPDP(MEM)=@LC(MEM)(ACLIST))
                  ACLIST(LC(MEM)=LC(MEM)+1)
(29)     ACLIST   -->   NULL
(30)     ACTT -->  STMNT('COMMENT',SENTT=ACTT)    ACT(%ACTT)
(31)     ACT(¬PRED,'ARRIV'|'ENTER')   -->
                  STMNT('GENERATE',XMOD=@MOBENATR(ACT)(ACT),ARGA=IETM(ACT))
                  STMNT('ASSIGN',XMOD=2,ARGA=1,ARGB=ASNDISTR(ACT),¬ARGB,
                        ARGB=@MOBENATR(ACT)(ACT))
                  STMNT('ENTER',ARGA=LOCOBJ(LOCATION(ACT)))
                  SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(32)     ACT($'ACTIVITY', STORIND($STRUC(LOCOBJ(LOCATION))))   -->
                  STMNT('QUEUE',LABL=ACT,ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('ENTER',ARGA=LOCOBJ(LOCATION(ACT)),
                        ARGB=CONSUMP($STRUC(@MOBENATR(ACT)(ACT))),
                        NUM(ARGB).EQ.1,-ARGB)
                  STMNT('DEPART',ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('ADVANCE',ARGA=DURATION(ACT))
                  STMNT('LEAVE', ARGA=LOCOBJ(LOCATION(ACT)),
                        ARGB=CONSUMP($STRUC(@MOBENATR(ACT)(ACT))),
                        NUM(ARGB).EQ.1,-ARGB)
                  SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(33)     ACT($'ACTIVITY')   -->
                  STMNT('QUEUE',LABL=ACT,ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('SEIZE',ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('DEPART',ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('ADVANCE',ARGA=DURATION(ACT))
                  STMNT('RELEASE',ARGA=LOCOBJ(LOCATION(ACT)))
                  SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
```

```
(34)      ACT('GOTO')    -->
                  STMNT('SELECT',LABL=ACT,MOD="TMIN",ARGA=NUM(ARGAZ(ACT)),
                  ARGB(ARGAZ(ACT)),ARGC(ARGAZ(ACT)),
                  ARGE(ARGAZ(ACT)))
                  SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(35)      ACT('LEAV',¬SUCC)    -->
                  STMNT('TABULATE',LABL=ACT,ARGA='TRANSREC',
                        ARGB='PARAM1',ARGC=0-1)
                  STMNT('LEAVE', ARGA=LOCOBJ(LOCATION(ACT)))
                  STMNT('TERMINAT')
(36)      ACT    -->    NULL
(37)      SUCSTMNT($'QTYP')    -->
                  STMNT('TEST',MOD="L",ARGA=%'SNAREC',NAM(ARGA)="Q",
                        NAM2(ARGA)=SUCARG(SUCSTMNT),ARGB=MAXQ(SUCSTMNT),
                        ARGC=CLOSACT(SUCSTMNT))
                  STMNT('TRANSFER',ARGB=OPENACT(SUCSTMNT))
(38)      SUCSTMNT($'FTYP'|$'STYP')    -->
                  STMNT('GATE',MOD="SNF",ARGA=SUCARG(SUCSTMNT),
                        ARGB=CLOSACT(SUCSTMNT),
                        ¬STORIND($STRUC(SUCARG(SUCSTMNT))),MOD="NU")
                  STMNT('TRANSFER',ARGB=OPENACT(SUCSTMNT))
          SUCSTMNT    -->    STMNT('TRANSFER',ARGB=IPDP(SUCSTMNT))
                  STMNT('COMMENT',CHARS=" TIMING LOOP",-IPDP(MEM))
**************************************************************************/
(40)      TMLOOP    -->
                  STMNT('COMMENT',CHARS=" TIMING LOOP")
                  STMNT('GENERATE',ARGA=PROBTIME(MEM))
                  STMNT('TERMINAT',ARGA=1)
                  STMNT('START',ARGA=1)
                  STMNT('END')
**************************************************************************/
(41)      STMNT(¬STORIND($STRUC(ARGA)), 'ENTER'|'LEAVE')    -->    NULL
(42)      STMNT('TRANSFER',ARGB.EQ.@LC(MEM)(ACPTR(MEM)))    -->    NULL
(43)      STMNT(ARGA$'UNIFORM')    -->
                  STMNT(ARGB=RANGE(ARGA),ARGA=MEAN(ARGA))
(44)      STMNT(ARGA$'EXPON')    -->
                  STMNT(ARGA=MEAN(ARGA),ARGB=%'SNAREC',NAM(ARGB)="FN",
                        NUM(ARGB)=1)
(45)      STMNT(ARGA$'COND')    -->
                  STMNT('GATE',TARGA=ARGA,
                        MOD=SMOD($(TARGA)),ARGA=CONDENTY(TARGA),
                        ¬STORIND($STRUC(ARGA)),MOD=FMOD($(TARGA)),-TARGA)
(46)      STMNT    -->    GSTMNT(NOCOPY,GPSSSW(MEM).EQ.1,%STMNT)
                  XSTMNT(NOCOPY,XVSW(MEM).EQ.1,%STMNT)
```

```
*************************************************************************/
*************************************************************************/
LEXOLOGY FOR ENCODING GPSS:
 (47)     GSTMNT(NOCOPY)     -->    NULL
 (48)     GSTMNT(XYLAST)     -->    XYOUT(%GSTMNT)
          GSTMNT('STORAGE',ARGA.EQ.1)     -->     NULL
 (50)     GSTMNT('TABULATE',ARGC)     -->
                GSTMNT(ARGA=ARGB,ARGB=ARGC,-ARGC,ARGB.LT.0,-ARGB)
 (51)     GSTMNT('COMMENT')     -->     NEWLINE1  *  NEWLINE1  *  COLUMN7
                COMMENT(%GSTMNT)
 (52)     COMMENT(CHARS)     -->     PHRASE(CHARS(COMMENT))
 (53)     COMMENT(SENTT)     -->
                SENT(%SENTT(COMMENT),-PRED,-SUCC,-CONDITN,-IETM,
                     -DURATION,LOCATION=%LOCATION,
                     LOCOBJ(LOCATION)=%LOCOBJ(LOCATION),
                     -LOCATION(LOCOBJ(LOCATION)))
 (54)     GSTMNT   -->    NEWLINE2     LABFLD(LABL(GSTMNT))
                          COLUMN8      OPFLD(SUP(GSTMNT))
                          COLUMN13     MODFLD(MOD(GSTMNT))
                          COLUMN19
                               ARGFLD(APTR=ARGA(GSTMNT),COMCNT(MEM)=0)
                               ARGFLD(APTR=ARGB(GSTMNT),NCOM=1)
                               ARGFLD(APTR=ARGC(GSTMNT),NCOM=2)
                               ARGDH(%GSTMNT)
 (55)     ARGDH('RMULT'|'SELECT'|'TABLE')     -->
                               ARGFLD(APTR=ARGD(ARGDH),NCOM=3)
                               ARGFLD(APTR=ARGE(ARGDH),NCOM=4)
                               ARGFLD(APTR=ARGF(ARGDH),NCOM=5)
                               ARGFLD(APTR=ARGG(ARGDH),NCOM=6)
                               ARGFLD(APTR=ARGH(ARGDH),NCOM=7)
 (56)     ARGDH   -->   NULL

 (57)     LABFLD(LABL)     -->     ARG(DATA=LABL(LABFLD))
 (58)     LABFLD     -->     NULL
 (59)     MODFLD(MOD)     -->     ARG(DATA=MOD(MODFLD))
 (60)     MODFLD     -->     NULL
 (61)     ARGFLD(APTR)     -->     COMMAS(NCOM(ARGFLD))  ARG(DATA=APTR(ARGFLD))
 (62)     ARGFLD     -->     NULL

 (63)     XYOUT     -->     NEWLINE1
                LINE(%XYOUT,LAST=LC+7,TLAST=XYLAST-LC,
                     TLAST.LT.7,LAST=LC+TLAST)
 (64)     LINE(LC.LE.LAST)     -->
                ARG(DATA=%'DECREC',NUM(DATA)=@LC(LINE)(LINE),
                    LC(LINE)=LC(LINE)+1,¬EXPNORM(LINE),
                    DATA=NUM(DATA))     ,
                ARG(DATA=%'DECREC',NUM(DATA)=@LC(LINE)(LINE),¬EXPNORM(LINE),
                    DATA=NUM(DATA))   /    LINE(LC=LC+1)
```

```
 (65)      LINE(LC.LE.XYLAST)     -->     XYOUT(%LINE)
 (66)      LINE     -->     NULL
*****************************************************************************/
*****************************************************************************/
MORPHOLOGY FOR ENCODING GPSS:
 (67)      OPFLD('TERMINAT')     -->     NAME(CHARS="TERMINATE")
 (68)      OPFLD('FVARIABL')     -->     NAME(CHARS="FVARIABLE")

 (69)      ARG(@9=DATA,GETYPE,@9.EQ.3)     -->     PARG(%DATA(ARG))
 (70)      ARG(@9.EQ.0)     -->     NUMBER(NUM=DATA(ARG))
 (71)      ARG     -->     NAME(CHARS=DATA(ARG))

 (72)      PARG($'ACTION')     -->     A     C     T     NUMBER(NUM=IDNO(PARG))
 (73)      PARG($'ENTITY')     -->     NAME(CHARS=IDNAME(PARG))
 (74)      PARG('DECIMAL')     -->     DECNUMB(NUM(PARG))
 (75)      PARG($'ABSTIME')     -->     TIMARG(%PARG)
 (76)      PARG($'QUANVAL')     -->     NUMBER(NUM(PARG))
 (77)      PARG('SNAREF',NUM)     -->
                NAME(CHARS=NAM(PARG))
                NUMBER(NUM(PARG))
 (78)      PARG('SNAREF')     -->     NAME(CHARS=NAM(PARG))     $
                NAME(CHARS=IDNAME(NAM2(PARG)))
 (79)      PARG('PARAMNO')     -->     P     NUMBER(NUM(PARG))
 (80)      PARG('TRANSTIM')     -->     M     1
 (81)      PARG('RANDM')     -->
                R     N     NUMBER(GRNNO(MEM)=GRNNO(MEM)+1,NUM=GRNNO(MEM),
                                     GRNNO(MEM).EQ.8,-GRNNO(MEM))
 (82)      PARG('NORMAL')     -->     V     NUMBER(NUM=IDNO(PARG))
 (83)      PARG(MEAN,STDEV)     -->     ARG(DATA=MEAN(PARG))     +
                ARG(DATA=STDEV(PARG))     *     F     N     2
 (84)      PARG(XYLAST)     -->     F     N     NUMBER(NUM=IDNO(PARG))
 (85)      PARG(NOPTS)     -->     NAME(CHARS=FNTYPE(PARG))
                NUMBER(NUM=NOPTS(PARG))
 (86)      COMMAS(NCOM.GT.COMCNT(MEM))     -->
                ,     COMMAS(COMCNT(MEM)=COMCNT(MEM)+1)
 (87)      COMMAS     -->     NULL
 (88)      NEWLINE1     -->     OUTPUT(@11=1,@12=1)
 (89)      NEWLINE2     -->     OUTPUT(@11=1,@12=2)
 (90)      NEWLINE8     -->     OUTPUT(@11=1,@12=8)
 (91)      COLUMN7     -->     OUTPUT(@12=7)
 (92)      COLUMN8     -->     OUTPUT(@12=8)
 (93)      COLUMN13     -->     OUTPUT(@12=13)
 (94)      COLUMN19     -->     OUTPUT(@12=19)
 (95)      NAME     -->     OUTPUT(@13=CHARS(NAME))
 (96)      NUMBER(¬NUM)     -->     0
 (97)      NUMBER     -->     OUTPUT(@14=NUM(NUMBER))
 (98)      DECNUMB(¬NUM)     -->     0.0
 (99)      DECNUMB     -->     OUTPUT(@15=NUM(DECNUMB))
```

```
**************************************************************************/
**************************************************************************/
LEXOLOGY FOR ENCODING X-VECTOR:
(100)     XSTMNT(NOCOPY|'COMMENT'|'EQU'|'SIMULATE')     -->     NULL
(101)     XSTMNT('SEIZE')     -->     XSTMNT('ENTER')
(102)     XSTMNT('RELEASE')     -->     XSTMNT('LEAVE')

(103)     XSTMNT('RMULT')     -->     XARGAC(%XSTMNT,FX(MEM)=3,-INDX(MEM))

(104)     XSTMNT('STORAGE')     -->     STORALL(%XSTMNT)     QALL(%XSTMNT)

(105)     STORALL     -->
                  SETDIR(DIR='STORDIR',LABL(STORALL))
                  XVECTOR(INDX=2,RIGHT=ARGA(STORALL))
                  XVECTOR(INDX=10,RIGHT=IPDP(STORALL))
                  NXTALPTR(INDX=10)
(106)     QALL     -->
                  SETDIR(DIR='QDIR',LABL(QALL))
                  XVECTOR(INDX=8,LEFT=IPDP(QALL))
                  NXTALPTR(INDX=8)
(107)     XSTMNT('TABLE')     -->
                  SETDIR(DIR='TABDIR',LABL(XSTMNT))
                  XVECTOR(INDX=6,RIGHT=ARGC(XSTMNT))
                  XVECTOR(INDX=8,RIGHT=ARGD(XSTMNT))
                  XVECTOR(INDX=9,RIGHT=ARGB(XSTMNT))
                  NXTALPTR(INDX=9+ARGD(XSTMNT))
(108)     NXTALPTR     -->     NULL(FX(MEM)=FX(MEM)+INDX(NXTALPTR))

(109)     XSTMNT('FUNCTION')     -->     SETDIR(DIR='FNDIR',LABL(XSTMNT))
                  XARG(DATA=ARGA(XSTMNT),INDX(MEM)=1)
                  XCODE(INDX=2,LEFT=23,RIGHT=IPDP(XSTMNT))
                  XCODE(INDX=4,LEFT=25)
                  XVECTOR(INDX=5,RIGHT=NOPTS(ARGB(XSTMNT)),LEFT=1,
                        FNTYPE(ARGB(XSTMNT)).NE."C",LEFT=2)
(110)     XSTMNT(XYLAST)     -->
                  VALTYPE(%XSTMNT,INDX(MEM)=6)
                  XXYOUT(%XSTMNT,TEMP=XYLAST-100,NOPTS=TEMP/2,XALL=7)

(111)     XXYOUT(LC.LE.XYLAST)     -->
                  XARG(DATA=%'DECREC',NUM(DATA)=@LC(XXYOUT)(XXYOUT),
                        LC(XXYOUT)=LC(XXYOUT)+1,INDX(MEM)=XALL(XXYOUT),
                        ¬EXPNORM(XXYOUT),DATA=NUM(DATA))
                  NXARG(%XXYOUT)
                  XXYOUT(LC=LC+1,XALL=XALL+1)
(112)     NXARG     -->
                  XARG(DATA=%'DECREC',INDX(MEM)=XALL(NXARG)+NOPTS(NXARG),
                        NUM(DATA)=@LC(NXARG)(NXARG),¬EXPNORM(NXARG),
                        DATA=NUM(DATA))
```

```
(113)     XXYOUT     -->
                  NULL(INDX=2*NOPTS(XXYOUT),INDX=INDX+6,
                       FX(MEM)=FX(MEM)+INDX,DORC(XXYOUT).EQ."C",
                       FX(MEM)=FX(MEM)+NOPTS(XXYOUT))
(114)     VALTYPE(@101$'DECIMAL'|EXPNORM)     -->
                  YCHK(%VALTYPE,LEFT=1)
(115)     VALTYPE     -->     YCHK(%VALTYPE)
(116)     YCHK(@102$'DECIMAL'|EXPNORM)     -->     XVECTOR(LEFT(YCHK),RIGHT=1)
(117)     YCHK     -->     XVECTOR(LEFT(YCHK))
(118)     XSTMNT('FVARIABLE')     -->
                  SETDIR(DIR='VARDIR',LABL(XSTMNT),INCREM=1,-INDX(MEM))
                  XARGFLD(APTR=ARGA(XSTMNT))

(119)     XSTMNT('START')     -->     XVECTOR(IX=1,RIGHT=1)
                  XVECTOR(IX=2,RIGHT=ARGA(XSTMNT))
(120)     XSTMNT('END')     -->
                  PTRALL(%'STORDIR',-INDX(MEM),FX(MEM)=FX(MEM)+1)
                  PTRALL(%'QDIR')
                  PTRALL(%'TABDIR')          PTRALL(%'FNDIR')
                  PTRALL(%'VARDIR')          PTRALL(%'BLKDIR')
                  TRANSPTR     STUFBACK(%'BACKSTUF',LC=11)
(121)     PTRALL     -->
                  XVECTOR(IX=DIRPTR(PTRALL),RIGHT=FX(MEM)-1)
                  XVECTOR(IX=DIRPTR(PTRALL)-1,RIGHT=LR(PTRALL)-10)
                  PTRDIR(%PTRALL,LC=11)
(122)     PTRDIR(LC.LE.LR)     -->
                  XVECTOR(RIGHT=@LC(PTRDIR)(PTRDIR))
                  PTRDIR(LC=LC+1)
(123)     PTRDIR     -->     NULL
(124)     TRANSPTR     -->
                  XVECTOR(IX=24,RIGHT=3)
                  XVECTOR(IX=25,RIGHT=FX(MEM)-1)
(125)     STUFBACK(LC.LE.LR)     -->
                  XVECTOR(IX=@LC(STUFBACK)(STUFBACK),
                          LC(STUFBACK)=LC(STUFBACK)+1,
                          RIGHT=BLOKNO(@LC(STUFBACK)(STUFBACK)))
                  STUFBACK(LC=LC+1)
(126)     STUFBACK     -->     NULL

(127)     XSTMNT     -->
                  SETDIR(DIR='BLKDIR',INCREM=1,-INDX(MEM),LABL(XSTMNT).NE.0,
                         BLOKNO(IPDP(MEM))=LR('BLKDIR')-9)
                  XVECTOR(LEFT=BTCODE(SUP(XSTMNT)),RIGHT=IPDP(MEM))
                  XARGAC(%XSTMNT)

(128)     XARGAC     -->
                  XARGFLD(APTR=ARGA(XARGAC),SNACNT(MEM)=0)
```

```
                    XARGFLD(APTR=ARGB(XARGAC),NSNA=1)
                    XARGFLD(APTR=ARGC(XARGAC),NSNA=2)
                    XARGDH(%XARGAC)
                    XMODFLD(SUP(XARGAC),MOD(XARGAC),¬MOD,XMOD(XARGAC))
(129)     XARGDH('RMULT'|'SELECT')    -->
                    XARGFLD(APTR=ARGD(XARGDH),NSNA=3)
                    XARGFLD(APTR=ARGE(XARGDH),NSNA=4)
                    XARGFLD(APTR=ARGF(XARGDH),NSNA=5)
                    XARGFLD(APTR=ARGG(XARGDH),NSNA=6)
                    XARGFLD(APTR=ARGH(XARGDH),NSNA=7)
(130)     XARGDH    -->    NULL

(131)     XMODFLD('RMULT')      -->     NULL(FX(MEM)=32)
(132)     XMODFLD     -->
                    XCODE(LEFT=33,MOD(XMODFLD),RIGHT=2,MOD.NE."L",
                          MOD.NE."SF",MOD.NE."U",RIGHT=3,MOD.NE."SNF",
                          MOD.NE."NU",RIGHT=8,MOD.NE."TMIN",
                          RIGHT=XMOD(XMODFLD))
(133)     XARGFLD(APTR)     -->    SNAS(NSNA(XARGFLD))  XARG(DATA=APTR(XARGFLD))
(134)     XARGFLD     -->     NULL
(135)     SETDIR(¬LABL)     -->     SETDIR(LABL=LR(DIR)-9)
(136)     SETDIR     -->
                    NULL(%SETDIR,LABL=LABL+10,
                         @LABL(SEG)(DIR)=FX(MEM),FX(MEM)=FX(MEM)+INCREM,
                         LABL.GT.LR(DIR),LR(DIR)=LABL)
**************************************************************************/
**************************************************************************/
MORPHOLOGY FOR ENCODING X-VECTOR:
(137)     XARG(@9=DATA,GETYPE,@9.EQ.3)     -->
                    XPARG(%DATA(XARG),IPDP=DATA(XARG))
(138)     XARG(@9.EQ.0)     -->     XVECTOR(RIGHT=DATA(XARG))
(139)     XARG     -->     NULL


(140)     XPARG($'ACTION')     -->
                    XVECTOR(RIGHT=IPDP(XPARG),LR('BACKSTUF')=LR('BACKSTUF')+1,
                            @LR('BACKSTUF')('BACKSTUF')=FX(MEM)+INDX(MEM),
                            LR('BACKSTUF')=LR('BACKSTUF')+1,
                            @LR('BACKSTUF')('BACKSTUF')=RIGHT)
(141)     XPARG($'ENTITY')     -->     XVECTOR(RIGHT=IDNO(XPARG))
(142)     XPARG('DECIMAL')     -->
                    OUTPUT(@11=1,@12=5)
                    NUMBER(NUM=FX(MEM)+INDX(MEM))   #   #
                    DECNUMB(NUM(XPARG))   #   #
                    NULL(IX=FX(MEM)+INDX(MEM),XV=NUM(XPARG),
                         PUTPX,¬INDX(MEM),FX(MEM)=FX(MEM)+1)
(143)     XPARG($'ABSTIME')     -->     TIMARG(%XPARG,XVSW)
(144)     XPARG($'QUANVAL')     -->     XVECTOR(RIGHT=NUM(XPARG))
```

```
(145)   XPARG('SNAREF',NUM)     -->
                XCODE(RIGHT=NUM(XPARG),NAM(XPARG),LEFT=22,
                NAM.NE."FN",-LEFT)
(146)   XPARG('SNAREF')     -->
                XCODE(RIGHT=IDNO(NAM2(XPARG)),NAM(XPARG),LEFT=12,
                NAM.NE."Q",LEFT=13,NAM.NE."QA",-LEFT)
(147)   XPARG('PARAMNO')     -->     XCODE(LEFT=1,RIGHT=NUM(XPARG))
(148)   XPARG('TRANSTIM')     -->     XCODE(LEFT=3,RIGHT=1)
(149)   XPARG('RANDM')     -->
                XCODE(LEFT=26,XRNNO(MEM)=XRNNO(MEM)+1,RIGHT=XRNNO(MEM),
                XRNNO(MEM).EQ.8,-XRNNO(MEM))
(150)   XPARG('NORMAL')     -->     XCODE(LEFT=24,RIGHT=IDNO(XPARG))
(151)   XPARG(MEAN,STDEV)     -->
                XARG(DATA=MEAN(XPARG))     XARG(DATA=STDEV(XPARG))
                XCODE(LEFT=22,RIGHT=2)
                XCODE(LEFT=30,RIGHT=3)
                XCODE(LEFT=30,RIGHT=1)
                XCODE(LEFT=25,RIGHT=0)
(152)   XPARG(XYLAST)     -->     XCODE(LEFT=22,RIGHT=IDNO(XPARG))
(153)   XCODE     -->     XVECTOR(%XCODE,LEFT=0-LEFT)
(154)   XVECTOR(INDX)     -->     XVECTOR(INDX(MEM)=INDX,-INDX)
(155)   XVECTOR(¬IX)     -->
                XVECTOR(IX=FX(MEM)+INDX(MEM),
                ¬INDX(MEM),TEMP=0-LEFT,TEMP.NE.33,TEMP.NE.25,
                FX(MEM)=FX(MEM)+1)
(156)   XVECTOR     -->
                OUTPUT(@11=1,@12=5)
                NUMBER(NUM=IX(XVECTOR))     #     #
                NUMBER(NUM=LEFT(XVECTOR))     #     #
                NUMBER(NUM=RIGHT(XVECTOR))     #     #
                NULL(%XVECTOR,XV=LEFT,PUTLX,XV=RIGHT,PUTRX)
(157)   SNAS(NSNA.GT.SNACNT(MEM))     -->
                SNAS(FX(MEM)=FX(MEM)+1,SNACNT(MEM)=SNACNT(MEM)+1)
(158)   SNAS     -->     NULL(SNACNT(MEM)=SNACNT(MEM)+1)
(159)   TIMARG(¬SUP)     -->     TNUMBER(NUM(TIMARG),XVSW=XVSW(TIMARG))
(160)   TIMARG(SUP.EQ.SUP(TIMUNIT(MEM)))     -->
                TNUMBER(NUM=NUM(TIMARG)/NUM(TIMUNIT(MEM)),XVSW=XVSW(TIMARG))
(161)   TIMARG(SUP.EQ.UNITS(SUP(TIMUNIT(MEM))))     -->
                TIMARG(NUM=NUM(TIMARG)/NUM(SUP(TIMUNIT(MEM))),
                SUP(TIMUNIT(MEM)))
(162)   TIMARG     -->     TIMARG(NUM=NUM*NUM(SUP),SUP=UNITS(SUP))
(163)   TNUMBER(XVSW)     -->     XVECTOR(RIGHT=NUM(TNUMBER))
(164)   TNUMBER     -->     NUMBER(NUM(TNUMBER))
```

APPENDIX D

SUMMARY OF GPSS/X-VECTOR SEGMENT TYPES

Semology for encoding GPSS/X-Vector:

GPSSPROG----------Top level segment type.

Attributes: none.

Function: Initiates the GPSS/X-VECTOR rules. Creates initial GPSS, SETAGL, STMNT, SELIST, EXPFUNC, NORMFUNC, DISTLIST, SUCLIST, DSTLIST, ACLIST, and TMLOOP segments, or goes to null.

INITIALGPSS-------Created from GPSSPROG.

Attributes: none.

Function: Initializes GPSS and X-VECTOR random number counters to 0, sets to 10 the value of the LR attribute of records that will store pointers to X-Vector entity allocations.

SELIST------------Created from GPSSPROG.

Attributes: Those of STALIST plus LC.

Function: Creates STENTITY, a copy of a STATENTY record, for further processing or goes to null.

STENTITY----------Created from SELIST.

Attributes: Those of a specific stationary entity record plus IPDP.

Function: Creates EQUCARD and a STMNT segment for a STORAGE. A pointer to the IPD record is maintained as the value of attribute IPDP.

MELIST----------Created from GPSSPROG.

Attributes: Those of MOBLIST plus LC.

Function: Creates MENTITY, a copy of a MOBENTY record for further processing, or goes to null.

MENTITY----------Created from MELIST.

Attributes: Those of the segment from which it was created.

Function: Creates two copies of itself as EQUCARD and TABLECARD for further processing.

TABLECARD---------Created from MENTITY.

Attributes: Those of a MOBENTY record.

Function: Creates a STMNT segment for a TABLE or goes to null.

EQUCARD----------Created from STENTITY and MENTITY.

Attributes: Those of a STATENTY or MOBENTY record.

Function: Creates STMNT for an EQUCARD.

EXPFUNC----------Created from GPSSPROG

Attributes: None.

Function: Creates FNDEF which is a copy of 'EXPONREC' and adds attribute IPDP, a pointer to 'EXPONREC', or goes to null.

NORMFUNC----------Created from GPSSPROG.

Attributes: None.

Function: Creates FNDEF which is a copy of 'NORMREC' and adds attribute IPDP, a pointer to 'NORMREC', or goes to null.

DISTLIST----------Created from GPSSPROG.

Attributes: Those of DSTRLIST plus LC

Function: Creates FNDEF, a copy of TYPDIST, TYPTABL, EXPON, NORMAL, or UNIFORM for further processing, or goes to null.

SUCLIST----------Created from GPSSPROG.

Attributes: Those of SCSRLIST plus LC.

Function: Creates FNDEF, a copy of a FRACTNL, PTYP, QTYP, FTYP, or STYP successor descriptor for further processing or goes to null.

FNDEF----------Created from EXPFUNC, NORMFUNC, DISTLIST, SUCLIST.

Attributes: Those of the segment from which it was created.

Function: Creates 2 segments that represent a FUNCTION and FUNCTION X-Y coordinates or goes to null.

DSTLIST----------Created from GPSSPROG.

Attributes: Those of DSTRLIST plus LC.

Function: Creates DISTDEF, a copy of TYDIST, TYPTABL, EXPON, NORMAL, or UNIFORM for further processing, or goes to null.

DISTDEF----------Created from DSTLIST

Attributes: Those of the segment from which it was created.

Function: For each record with a SUP of 'NORMAL' it creates a STMNT segment for FVARIABLE, or goes to null.

ACLIST----------Created from GPSSPROG.

Attributes: Those of ACTNLIST.

Function: Creates ACTT, a copy of an EVENT or ACTIVITY for further processing, or goes to null. Adds IPDP which points to the action record processed.

ACTT----------Created from ACLIST.

Attributes: Those of the action record from which it was created.

Function: Creates a STMNT for a comment card and an ACT segment.

ACT----------Created from ACTT.

Attributes: Those of the action record which has been copied down to this level.

Function: Creates a STMNT for each block required for the action.

SUCSTMT----------Created from ACT.

Attributes: Those of successor descriptor records or an action record.

Function: Creates segments for TEST, TRANSFER, or GATE blocks from successor segments.

TMLOOP----------Created from GPSSPROG.

Attributes: None.

Function: Creates STMNT segments for the timing loop blocks and START and END.

STMNT----------Created from GPSSPROG, STENTITY, EQUCARD, TABLECARD, FNDEF, DISTDEF, ACT, SUCSTMNT, TMLOOP, STMNT.

Attributes: SUP, LABL, MOD, ARGA, through ARGH

Functions: Preprocesses certain segments that have no further use and those whose arguments need amplification to create either a NULL segment or another STMNT segment. If no special conditions exist on the left, creates copies of itself as GSTMNT and/or XSTMNT segments.

Lexology for encoding GPSS:

GSTMNT(Condition)-Created from STMNT.

Attributes: SUP, LABL, MOD, ARGA through ARGH.

Function: Preprocesses segments that are not applicable in GPSS and go to null or that require other than normal processing.

COMMENT----------Created from GSTMNT with a sup of 'COMMENT'.

Attributes: SUP, CHARS or SENTT.

Function: Create PHRASE or SENTT segments to insert comments in the GPSS program.

GSTMNT(Default)---Created from STMNT.

Attributes: SUP, LABL, MOD, ARGA through ARGH.

Function: Creates a series of segments that represent the fields in a GPSS statement and a segment ARGDH for further processing. ARGFLD segments are created for the first 3 argument attributes of the segment.

ARGDH------------Created from GSTMNT (Default)

Attributes: Those used are SUP and ARGD through ARGH.

Functions: Creates ARGFLD segments for ARGD through ARGH or goes to null.

LABFLD----------Created from GSTMNT (Default)

Attributes: LABL

Function: Creates ARG with an attribute DATA with the value of LABL.

MODFLD----------Created from GSTMNT (Default)

Attributes: MOD

Function: Creates an ARG with an attribute DATA with the value of MOD.

ARGFLD----------Created from GSTMNT (Default)

Attributes: APTR, NCOM

Function: Created COMMAS and ARG segments representing a comma and the value of an argument.

XYOUT----------Created from GSTMNT (XYLAST)

Attributes: Those of the segment from which it was created plus LAST.

Function: Creates NEWLINE1 that starts a new line in column 1 and a LINE segment for further processing.

LINE----------Created from XYOUT

Attributes: LC, LAST, XYLAST, attributes 101 through the value of XYLAST.

Function: Initiates output of GPSS Function Follower card X-Y pair values by creating two ARG segments each with attribute DATA containing the value of one of the pair or a pointer to the record 'DECREC' with attribute NUM taking on the value of Exponential or Normal distribution X-Y value.

The value of LC is compared to the value of LAST so only 4 pairs of coordinates are output. When the value of LC is greater than XYLAST, output is complete.

Morphology for encoding GPSS:

OPFLD----------Created from GSTMNT (Default)

Attributes: SUP

Function: To enable the output of the nine letter strings "terminate" and "FVARIABLE".

ARG----------Created from LABFLD, MODFLD, ARGFLD, LINE

Attributes: DATA

Function: Segments are created depending on the type of the value of DATA. Segments created may be PARG, NUMBER, or NAME.

PARG----------Created from ARG.

Attributes: Those of the segment from which it was created.

Function: Create segments whose type depends on the condition on the left that is satisfied. The created segments cause an output of single letters for letter segments, character string for NAME segment, and integer or decimal values for NUMBER or DECNUMB segments.

COMMAS----------Created from XARGFLD

Attributes: NCOM

Function: Causes the issue of commas prior to each GPSS argument or fills in commas for any missing arguments, or goes to null.

NEWLINE1----------Created from GSTMNT with a sup of 'COMMENT'

Attributes: None.

Function: Starts a new line in column 1.

NEWLINE2----------Created from GSTMNT (Default)

Attributes: None.

Function: Starts a new line in column 2.

NEWLINE8----------Created from GSTMNT (Default)

Attributes: None.

Function: Starts a new line in column 8.

COLUMN8-----------Created from GSTMNT (Default)

Attributes: None.

Function: Moves the column pointer to column 8.

COLUMN13----------Created from GSTMNT (Default)

Attributes: None

Function: Moves the column pointer to column 13.

COLUMN19----------Created from GSTMNT (Default)

Attributes: None.

Function: Moves the column pointer to column 19.

NAME--------------Created from ARG, various PARG segments

Attributes: CHARS

Function: Prints the EBCDIC value of CHARS.

NUMBER------------Created from ARG, various PARG segments.

Attributes: NUM

Function: Prints the integer value of NUM.

DECNUMB----------Created from PARG with a sup of 'DECIMAL'.

Attributes: NUM

Function: Prints the decimal value of NUM (considered to be in parts per thousand

Lexology for encoding X-Vector:

XSTMNT(Condition) Created from STMNT, XSTMNT.

Attributes: SUP, LABL, MOD, XARGA through XARGH.

Function: Segments representing action records are processed to initiate action creating entity allocations and store pointers to the allocation in 'BLKDIR'.

XSTMNT(Default)---Created from STMNT

Attributes: SUP, LABL, MOD, XARGA through XARGH

Function: A segment with NOCOPY, 'COMMENT', 'EQU', or 'SIMULATE' conditions is not applicable to X-Vector and goes to null. A segment with a SUP of 'SEIZE', or 'RELEASE' is changed to one with a SUP of 'ENTER' or 'LEAVE' as X-Vector only recognizes storages. A segment with a SUP of 'STORAGE' creates segments representing a STORAGE and QUEUE segments with a SUP condition create segments that represent blocks with the same name as the SUP. A segment with an attribute of XYLAST creates segments initiating the output of the FUNCTION follower card with X-Y coordinates.

STORALL----------Created from XSTMNT with a SUP of 'STORAGE'.

Attributes: Those of a XSTMNT segment with a SUP of 'STORAGE'.

Function: Segments representing STORAGES are processed to create a STORAGE allocation in 'STORDIR'.

QALL--------------Created from XSTMNT with a SUP of 'STORAGE'.

Attributes: Same as STORALL.

Function: Same as the STORALL function only for QUEUES.

NXTALPTR----------Created from STORALL, QALL, and XSTMNT with a SUP of 'TABLE'.

Attributes: INDX.

Function: Updates the master X-Vector pointer, attribute FX of MEMORY, to point to the last element of allocations for STORAGES, QUEUES, or TABLES.

XXYOUT------------Created from itself or XSTMNT with an attribute of XYLAST.

Attribute: Those of the segment from which it was created plus NOPTS.

Function: Creates an XARG segment to assign the X value of the X-Y pair to the attribute DATA and a segment representing the value of the Y value.

NXARG-------------Created from XXYOUT.

Attributes: Those of the segment XXYOUT from which it was created.

Function: Creates an XARG segment to assign the Y value of the X-Y pair to the attribute DATA.

VALTYPE----------Created from XSTMNT with an attribute of XYLAST.

Attributes: All attributes of the XSTMNT segment from which it was created.

Function: Outputs a 1 to the left half of an X-Vector element if the X value of a pair is a decimal.

YCHK-------------Created from VALTYPE

Attributes: All attributes of the XSTMNT segment from which it was created.

Function: Outputs a 1 to the right half of an X-Vector element if the Y value is a decimal.

PTRALL-----------Created from XSTMNT with a SUP of 'END'.

Attributes: Those of the named record of which it is a copy. The named record is a list of pointers to specific entity allocations.

Function: Creates segments with attributes assigned the values of the number of a specific entity and a pointer to the directory allocation and creates PTRDIR.

PTRDIR-----------Created from PTRALL.

Attributes: Those of PTRALL from which it was created, plus LC.

Function: Creates a segment with attribute RIGHT assigned the value of a pointer to a specific allocation.

TRANSPTR----------Created from XSTMNT with a SUP of 'END'.

Attributes: None.

Function: The value of a pointer to the free storage area of X-Vector is put into attribute RIGHT, and its location, 25, in IX.

STUFBACK----------Created from XSTMNT with a SUP of 'END'.

Attributes: Those of 'BACKSTUF' plus LC.

Function: Causes the value of BLOKNO to be inserted into its proper X-Vector location.

XARGAC------------Created from XSTMNT with a SUP or 'RMULT'. XSTMNT (Default)

Attributes: SUP, LABL, MOD, XARG through XARGH

Function: Causes eventual output of the first 3 arguments of a Block allocation, seed or select allocation. Creates segments XARGDH and XMODFLD.

XARGDH------------Created from XARGAC.

Attributes: Primarily SUP and XARGD through XARGH

Function: Causes eventual output of the value of XARGD through XARGH.

XMODFLD-----------Created from XARGAC.

Attributes: SUP, MOD, or XMOD.

Function: Causes code -33 and code for MOD type to be entered in last element of a BLOCK allocation if XMOD attribute is presen .

Causes entry of pointer to original action record in right half of the last element.

XARGFLD----------Created from XARGAC, XARGDH, XSTMNT with a SUP of 'FVARIABLE'.

Attributes: APTR.

Function: Causes output of argument value codes to SNA elements of a Block allocation. If an argument is missing, an element is assigned, as an argument's position indicates which argument it is.

SETDIR----------Created from STORALL, QALL, XSTMNT with a SUP of 'TABLE', XSTMNT (Default)

Attributes: DIR, LABL, INCREM if SUP is 'FVARIABLE' or XSTMNT has no condition on the left.

Function: Saves pointers to the allocations for all entities in a record pointed to by the value of DIR.

Morphology for encoding X-Vector:

XARG----------Created from XSTMNT with a SUP of 'FUNCTION', XXYOUT, NXARG, and XARGFLD.

Attributes: DATA

Function: Segments are created depending on the type of the value of DATA, segments may be XPARG, NUMBER, or NULL.

XPARG----------Created from XARG

Attributes: Those of the segment from which it was created plus IPDP.

Function: Causes an output to the X-Vector primarily for each argument of a BLOCK allocation.

XCODE----------Created from XSTMNT with a SUP of 'FUNCTION', XMODFLD (No Conditions), XPARG segments with a SUP of 'SNAREF', 'PARAMNO', 'TRANSTIM', 'RANDM', 'NORMAL', or XPARG segments with attributes of MEAN and STDEV, XYLAST.

Attributes: LEFT, RIGHT, NAM from a segment with a SUP of 'SNAREF'.

Function: Causes a negative code value to be entered in the left half of an element as required for identification by the simulation routine.

XVECTOR----------Created from a variety of segments.

Attributes: INDX from various segments, IX from some segments, LEFT and RIGHT from all segments creating XVECTOR. This value may be 0.

SNAS----------Created from XARGFLD.

Attribute: NSNA

Function: Skips elements in Block allocations if that argument is not there.

TIMARG----------Created from XPARG in the set 'ABSTIME'.

Attributes: NUM, UNITS.

Function: Adjusts any times mentioned in the problem with units not the same as the basic time unit.

TNUMBER----------Created from TIMARG

Attributes: Indicator XUSW and attribute NUM or just NUM.

Function: Creates an XVECTOR segment if XUSW is set, otherwise a NUMBER segment is created.

LIST OF REFERENCES

1. Lamb, Sydney M., Outline of Stratificational Grammar, Georgetown University Press, Washington, D.C., 1966.

2. Heidorn, George E., Natural Language Inputs to a Simulation Programming System--An Introduction, Technical Report No. NPS-55HD71121A, Naval Postgraduate School, Monterey, California, December 1971.

3. Heidorn, George E., Natural Language Inputs to a Simulation Programming System, Technical Report, Naval Postgraduate School, Monterey, California, in preparation.

4. Baker, Eldon S., Question-Answer Inputs to a Simulation Program Generating System, M.S. Thesis, Naval Postgraduate School, June 1971.

5. McGee, Robert T., The Translation of Data Structure Representations of Simple Queuing Problems into GPSS Programs and English Text, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1971.

6. Hansen, Richard C., Ges: A Data-Structure-to-GPSS Encoding System, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1970.

7. General Purpose Simulation System/360 - Introductory User's Manual, Publication H20-0304, IBM DP Div, White Plains, N. Y., 1967.

8. Hemphill, Frederick H., Jr., Computer Verification of the Completeness of a Simulation Problem Description by Natural Language Interaction, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1971.

9. Williams, Robert J., A GPSS-Like Simulator Callable from a Fortran Program, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1972.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assistant Professor George E. Heidorn, Code 55HD Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	5
4.	LCDR J. H. Rickelman, USN COMSUBLANT Staff, Code N7 Norfolk, Virginia 23511	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

Translation of a Queuing Problem Description Into GPSS-Like Tables

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Master's Thesis; June 1972

5. AUTHOR(S) *(First name, middle initial, last name)*

John Henry Rickelman

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1972	96	9

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A system called NLPQ is being developed at the Naval Postgraduate School as part of a research project in natural language man-machine communication. The basic part of the system consists of a rule language and programs to compile and execute rules. NLPQ currently allows a user to enter an English text description of a simple queuing problem at a time sharing terminal, have the computer construct an Internal Problem Description, and then produce an equivalent English text description and a GPSS simulation program to solve the problem.

Recently, a FORTRAN routine for performing a GPSS-like simulation was incorporated into the system. This thesis reports on the development of another set of rules for NLPQ to translate an Internal Problem Description into an array of information required by this FORTRAN routine to perform the simulation.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Digital Computer Simulation						
Simulation Programming						
GPSS						
Computational Linguistics						
Grammar-Rule Language						
Stratificational Grammar						

DD FORM 1 NOV 65 1473 (BACK)

S/N 0101-807-6821

A-31409

141764

Thesis
R3957
c.1

Rickelman
Translation of a queuing problem description into GPSS-like tables.